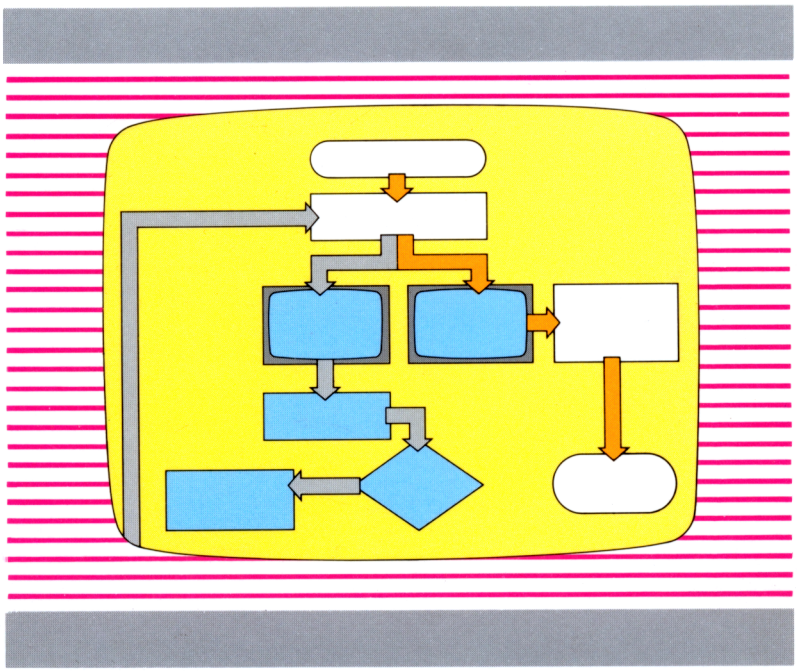
ATARI ST SERIES

Series Editor: Robin Bradbeer
Foreword by Jack Tramiel

USING ST BASIC ON THE ATARI ST



Dick Meadows

GLENTOP

USING ST BASIC ON THE ATARI ST

USING ST BASIC ON THE ATARI ST

Dick Meadows

GLENTOP PUBLISHERS LTD

First published 1986

©Newtech/Glentop Publishers Ltd, 1986
World rights reserved

No part of this publication may be copied, transmitted, or stored in a retrieval system or reproduced in any way including, but not limited to, photograph, photocopy, magnetic or other recording means, without prior written permission from the publishers, with the exception of material entered and executed on a computer system for the reader's own use.

ISBN

1 85181 051 X

Printed and bound in Great Britain by
Paradigm Print, Gateshead, Tyne and Wear.

A Newtech production for
Glentop Publishers Ltd
Standfast House, Bath Place, Barnet, Herts EN5 5XE.

CONTENTS

Preface 13

Chapter 1 **Getting started with ST BASIC** 15
Introduction and summary 15
Loading ST BASIC 15
The PRINT command and calculations 20
Clearing the screen windows 23
Making a window full size 23
Further applications of PRINT 24
Drawing lines, circles and ellipses 28
Further calculations 36

Chapter 2 **Beginning to write and run programs** 39
Introduction and summary 39
What is a program? 39
Fundamental commands and statements 42
Writing and running simple programs 46
Variables and identifiers 50
Obtaining hard copy 53
Editing programs 53
Saving programs on disk 56

Loading programs from disk 58
The DIRectory command 58
Merging programs 59
Deleting programs stored on disk 59
Debugging programs 60

Chapter 3 **Interacting with your program: data input** **65**
Introduction and summary 65
INPUT statements 66
READ-DATA statements 69
STOP statement and CONtinue command 71
RESTORE statement with READ-DATA 72

Chapter 4 **Decision-making, repetition, jumping and subroutines** **76**
Introduction and summary 76
IF statements 77
Repetition: the FOR loop 80
Repetition: the WHILE loop 85
Boolean operators and logic expressions 87
GOTO commands, statements and use of labels 90
GOSUB statements and subroutines 94

Chapter 5 **Standard functions and applications** **98**
Introduction and summary 98
Square roots: SQR(X) 98
ABSolute, INTeger and SIGN of numbers 99
Random-number generation 100
Trigonometric functions 103
Exponential and logarithmic functions 106
User-defined functions 107
Strings and string slicing 109
Character sets and codes 112
Denary-to-hexadecimal conversion 114

Chapter 6 **Applications of arrays and files** **117**
Introduction and summary 117
Arrays and the DIM statement 117
Use and application of arrays 121
Sorting numbers in order 126
Sorting in alphabetical order 129
Using files in programs 131

Chapter 7 **Some practical programs** 137
Introduction and summary 137
Converting common measures 138
Currency conversions 140
Multiple copies and standard letters 142
Checking bills 144
Income tax calculation 147
Interest and return on investment 148
Loan repayment 150

Appendix 1 **Answers to exercise problems** 153

Appendix 2 **Summary of ST BASIC keywords** 165

Appendix 3 **ST BASIC reserved words** 182

Index 185

ILLUSTRATIONS

Figure

1.1 Basic ST Computer system 16
1.2 Display obtained when operating system is fully loaded 17
1.3 Display after "clicking on File" 17
1.4 Display with ST BASIC disk in drive unit and after "clicking" on the Open option of the File menu 18
1.5 Display when ST BASIC has been fully loaded 18
1.6 Screen display after execution of calculation command 21
1.7 Screen display after execution of second calculation command 21
1.8 Display obtained on running TAB illustration program 27
1.9 Coordinate system for graphics commands in output window 29
1.10 Output display produced by the graphics commands LINEF, CIRCLE and ELLIPSE 31
1.11 Drawing arc lengths 32
1.12 PCIRCLE and COLOR commands displays 34

1.13 Screen display produced by program illustrating the action of the FILL function 35

Figure

2.1 Area of triangle 48
2.2 Display for cylinder program 49
2.3 Selecting the Edit mode; screen appearance after clicking on Edit 54
2.4 Clicking on the Start Edit option to bring down the Edit window 54
2.5 The Help Edit list explaining the use of the F function keys 55
2.6 Saving programs on disk 57
2.7 Debug menu and Trace and Tron options 61
2.8 Figures for problems 63

Figure

4.1 Flowchart representation of IF...THEN and IF...THEN...ELSE statements 77
4.2 Flowchart representation of FOR loop 81
4.3 Display produced by concentric circle program 84
4.4 Flowchart representation of WHILE...WEND loop 86

Figure

5.1 Basic trigonometric functions 104
5.2 Sine rule 105
5.3 Cosine rule 106
5.4 Plot of exponential function e^x 107
5.5 Part of ASCII code character set 113

Figure

6.1 Pictorial representation of arrays 118
6.2 Pictorial representation of bubble-sort method 128

Figure

7.1 Conversion of units program 138
7.2 Conversion table for some common quantities 139
7.3 Currency exchange calculation 140
7.4 Programs for producing multiple copies, standard letters, etc. 142
7.5 A program for checking domestic bills 145

7.6 A program for income tax calculations 147
7.7 Programs to calculate returns on regular savings and lump-sum investment 149
7.8 A program for loan repayment calculations 151

FOREWORD

When we introduced the ST series of computers at Atari we coined the phrase "power without the price". This summed up all that had been in our minds when we decided to design a range of powerful but low cost machines that could be used for all applications ranging from sophisticated games to complex business and scientific uses.

During the past few years, ever since I was responsible for bringing the first mass-produced electronic calculators and then the first true computers to the public at an affordable price, my whole aim has been to bring the benefits of technology to those of average income. We have to get high technology out of the hands of the few into the hands of the many. As I've said before, we want "classes for the masses". If you give somebody some sophisticated machinery then you'll be surprised what they can do with it. Time and again we have been amazed at what users have done with the technology when it is made freely available at an affordable price.

And that brings me on to this series of books, edited by my old acquaintance Robin Bradbeer. It is impossible to give all the information necessary to completely cover all the uses of a computer in the instruction manual. Also, if more than one person explains something they bring out differing strengths of the system. This series of books should help all users of the ST to get to know the machine better and therefore use it more productively. Who knows, we at Atari may yet again be surprised by what you, the user, can do with the affordable technology that we have provided.

JACK TRAMIEL
7 February 1986

INTRODUCTION

When we first had the idea of a series of books based around the Atari ST series of computers it was rather like shooting in the dark. We could see that the system had great potential from a technical point of view. But could Atari deliver and would it live up to its paper promises? Instead of rushing into print immediately we decided to hold back and see what the ST system was capable of, and also what it couldn't do. This series of books is based around authors who know what they're doing and who have established themselves as respected writers of computer-based literature.

When I first met Jack Tramiel, some 12 years ago now, I was impressed by his vision of providing the highest technology to the average "man in the street". It fitted closely with my own concerns of bringing technology to the people, explaining it in simple terms and then letting them get on with it! We have been constantly surprised during the past two decades with what the average person will do when presented with, say a computer, the workings of which have been simply explained. Most of the applications software used by industry today has not come from established software houses but from individuals or small groups of, usually amateur, users who have wanted to use a computer for a specific purpose and found that there is nothing on the market for them. So they've written it themselves and then found that there are many thousands of others with the same needs.

Microsoft BASIC, Visicalc, and Compsoft DMS are typical examples. So was the Apple II computer! In putting this series together I have had these thoughts in the back of my mind. Starting from simply introductory books we go through applications-orientated books to very detailed software notes explaining how every single bit of the machine works. During the next year or so, more books will be added to the series making it one of the most comprehensive ever devised for a single machine from one publisher. It should be possible for the reader to start at the simplest level and as they get more familiar with the system to work their way through the series until they know everything there is to know about it.

Of course if you have any comments, or would like to fill any gaps, then please let us know!

ROBIN BRADBEER
April 1986

PREFACE

This book has been written as a practical guide to using ST BASIC on the Atari ST computers. No prior knowledge of computing is assumed, nor is it a prerequisite to follow the text. The principal aim of the book is to give both newcomers, as well as more experienced users, sufficient knowledge and understanding of the Atari to enable them to write their own BASIC programs. At all stages, new concepts are thoroughly explained and backed-up by practical program examples. Coverage extends from the basics of the Atari ST computers and their operation to easy-to-understand, but nevertheless powerful, programming techniques.

DICK MEADOWS
London, June 1986

CHAPTER 1

GETTING STARTED WITH ST BASIC

Introduction and summary

This introductory chapter is designed to introduce to the newcomer the Atari ST series of computers and to describe how to get started in using the powerful ST BASIC language. We explain first how to load the ST BASIC language stored on disk into the computer's memory and describe the function of the four basic windows we shall be using. We then straightaway learn how to communicate with the computer using one of the most important BASIC commands, PRINT. Using PRINT we see how to use the computer as a calculator, how to display characters, words, symbols, etc., and how to provide different formats for the display of results, etc., in the output window.

We also consider how windows may be cleared, altered in size and moved around. The graphics functions for drawing lines, circles and ellipses are also introduced together with the colour and fill commands.

Loading ST BASIC

Figure 1.1 shows the basic set up of the Atari ST computer system with disk drive, monitor, mouse and power units appropriately connected. Before we can use the system for ST BASIC we must first load the computer with the ST BASIC disk.

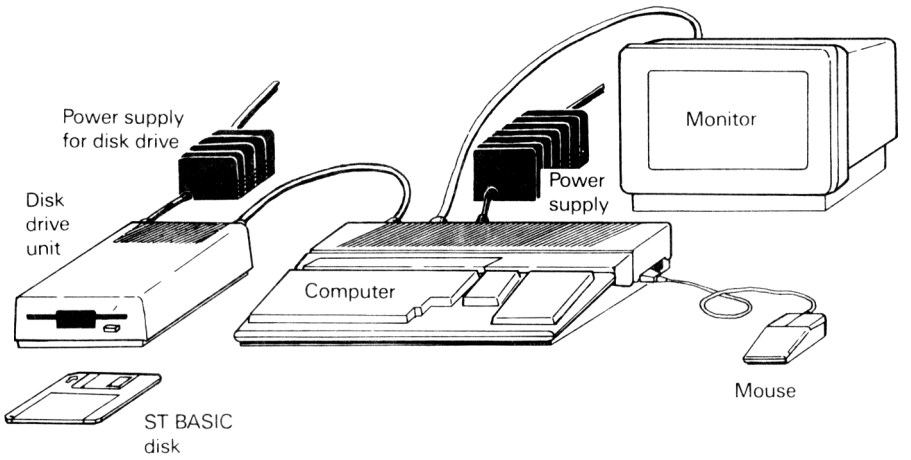


Figure 1.1 *The units making up a basic computer system*

The procedure is as follows:

- Switch on all units. Assuming that your computer has the operating system in ROM, i.e. permanently stored within the computer, just wait 50 or so seconds while the computer boots itself up. You are ready for the next stage as soon as the display of Figure 1.2 appears.

  In older versions of the ST series computers the operating system was not in ROM so for these cases you must first load the Systems Disk. Then when you switch on you are given simple loading instructions on the screen. Simply follow these and you will end up with the display of Figure 1.2.

- Place the ST BASIC language in the disk drive unit and move the mouse pointer so it is over the floppy disk A icon (the filing cabinet picture with A on it). Click the left-hand mouse button. The floppy disk A icon will become 'highlighted', i.e. the background or fill colour becomes black, the outlines and text become white.

- Now move the mouse pointer to File, second word on the top line (menu line) of the screen. File will become "highlighted" and the File menu option list will immediately drop down (see Figure 1.3).

- Move the mouse pointer to Open, the first item on the list. Open will also become "highlighted". Now click the left-hand mouse button.

- The disk unit operates and virtually immediately the list of files, programs, etc., contained on the ST BASIC list is displayed as shown in Figure 1.4.

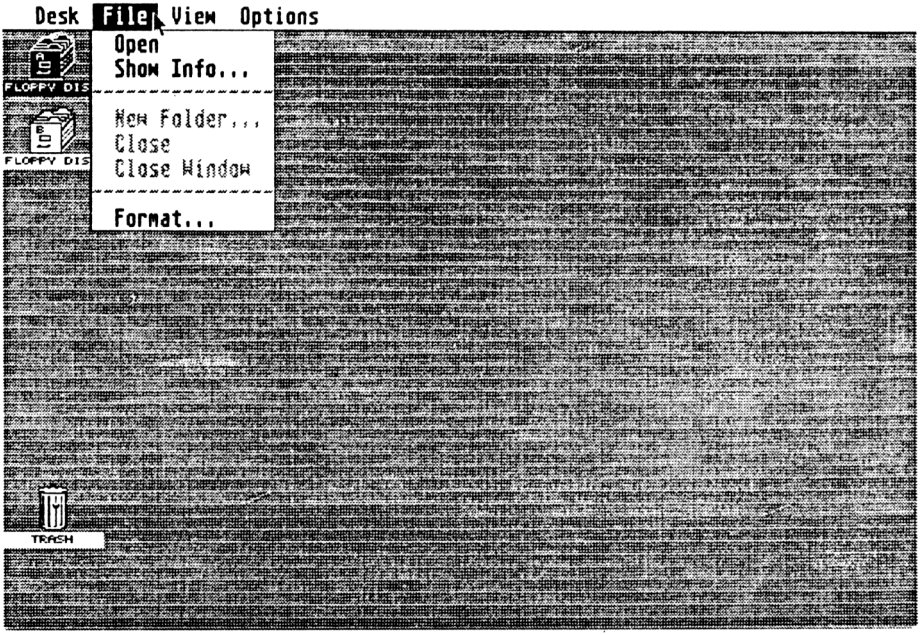


Figure 1.2 *Display obtaining when operating system is fully loaded (about 50 seconds after switch-on)*

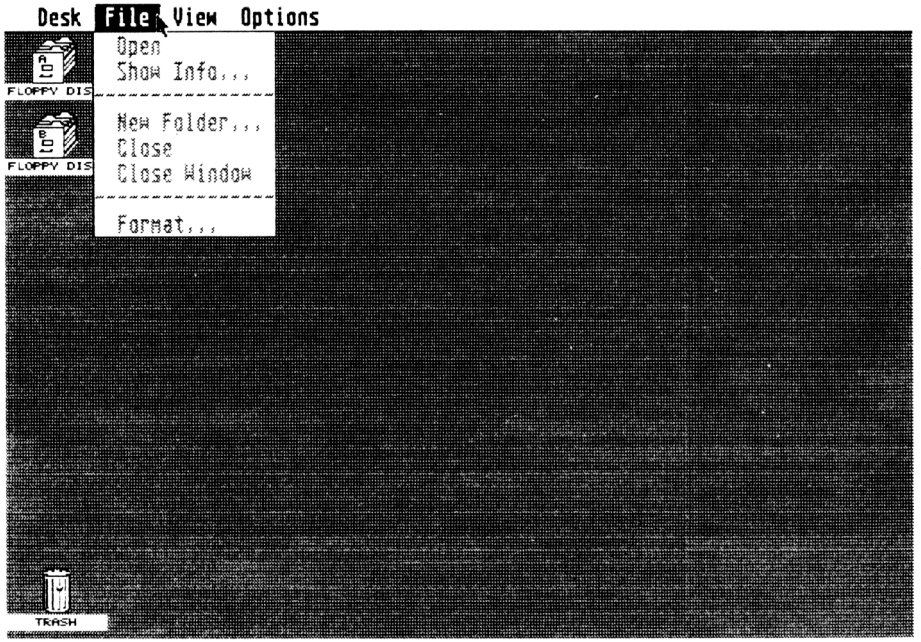


Figure 1.3 *Display after "clicking" on File*

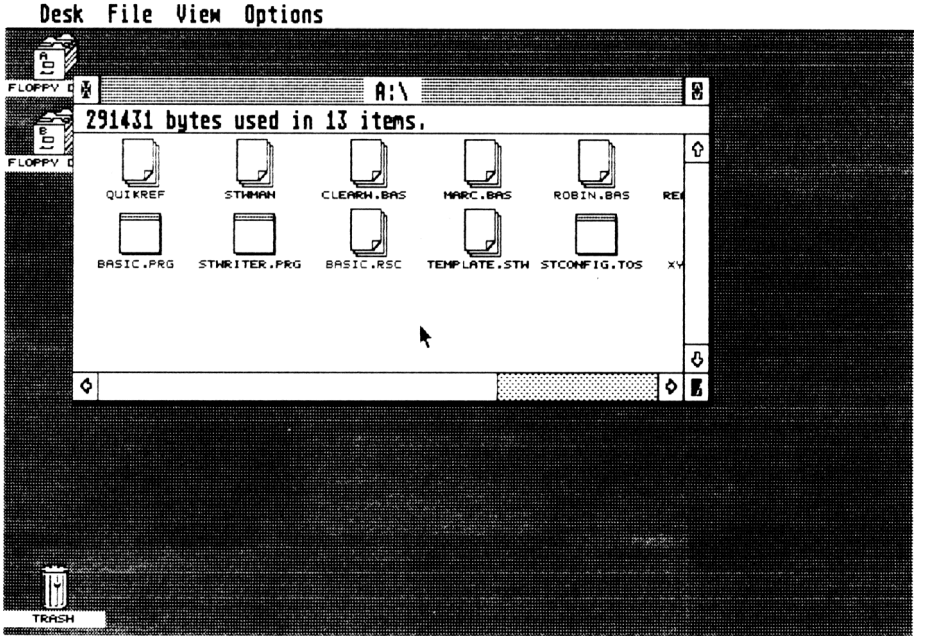


Figure 1.4 *Display with ST BASIC disk in drive unit and after "clicking" on the Open option of File menu*

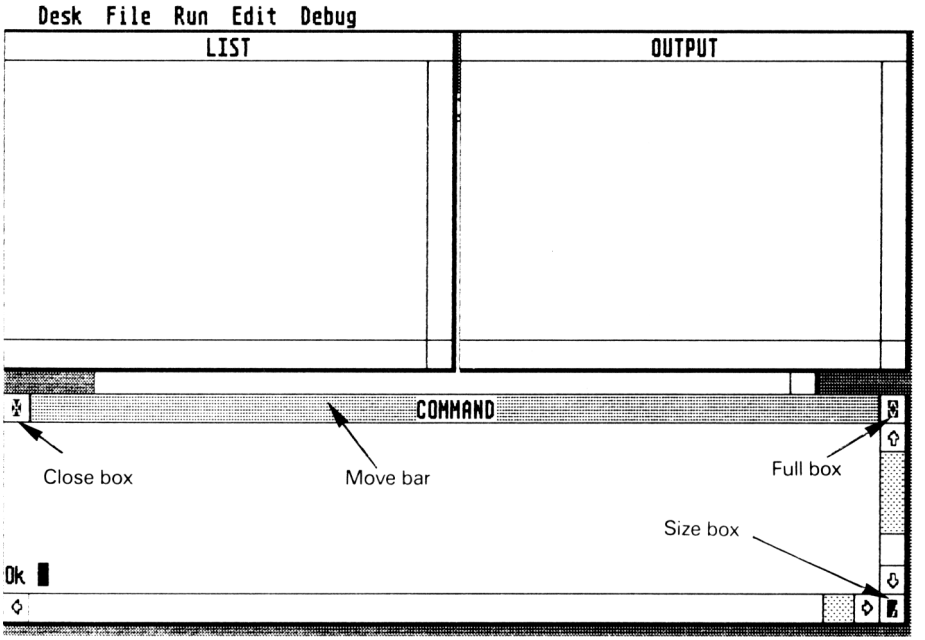


Figure 1.5 *Display when ST BASIC has been fully loaded; system is now ready to go*

- Move the mouse pointer over the BASIC.PRG icon and click the left-hand mouse button. The icon will become "highlighted".

- Now move the mouse pointer to File on the menu bar. The file list will drop down. Position the pointer on the Open option and click the left-hand mouse button once again.

- The ST BASIC is now loaded into the computer. The screen clears and the mouse pointer "changes" to a bee, indicating the computer is busy being loaded. Loading takes about 20 seconds. When loading is complete the BASIC desktop display of Figure 1.5 appears on the screen. You are now ready to go!

The initial desktop always displays three windows:

- The LIST window which is used to display the listing of the program currently stored in the computer

- The COMMAND window where our typed in instructions will appear

- The OUTPUT window where the output results obtained in running our program will be displayed.

A fourth window, not visible at this time, is the EDIT window. This "pops" up when we wish to edit our program, to make changes, etc. (see EDIT command, Chapter 2).

At the moment all windows are blank except for the command window which displays

Ok■

indicating the computer is ready to accept our commands. The small black square after Ok is the cursor. This will move to the right as you type in your commands and instructions.

Note: sizing and moving windows The windows can be made any size using the Size Box in the lower right-hand corner of an "active"window (a window is made active by moving the mouse pointer to lie within its area and clicking the left-hand button of the mouse – at the moment, the command window is active). Point the mouse arrow pointer on the size box, then press and hold down the left-hand mouse button. Then by dragging the Size Box along with the pointer you can alter the active window size. An active window can also be made "full-size" by clicking on the Full Box which is located at the top right-hand corner of the window or, as we shall see later, using the FULLW N command.

You can also move windows about the screen using the Move Bar which is located on the top edge of an active window. Locate the mouse pointer on the Move Bar, press and hold down the left-hand mouse button and drag the window to the required location.

For more details on sizing, moving and also for scrolling and opening/closing windows consult the Atari Owner's Manual.

The PRINT command and calculations

Here we explain how to use the computer as a calculator in its direct mode – "direct" because the computer acts on the command typed in directly and outputs the result to be displayed in the output window. To effect calculations we utilise the BASIC output command PRINT. All we need to do is to type in PRINT, in either upper or lower case, press the SPACE BAR to leave a space and then enter the calculation expression we wish to be evaulated using the number keys and the relevant arithmetic symbol keys, i.e.

+ key for addition
– key for subtraction
* key for multiplication
/ key for division

Your print command and calculation expression will be reproduced in the command window as you type it in. The calculation is performed and the result displayed in the output window immediately you press either the RETURN or ENTER keys. If you make any mistakes in entering these may be deleted by either using the DELETE or BACKSPACE keys or simply press RETURN and start again.

Examples

- To find 49 + 43 + 19 type in the following:

  print 49+43+19 [RETURN] or [ENTER]

  [RETURN] indicates that the RETURN key must be pressed, likewise [ENTER] indicates the ENTER key is to be pressed. Note either key may be pressed to execute the calculation.

  The command is displayed in the command window and as soon as the RETURN or ENTER key is pressed the answer 111 is displayed in the output window. Ok appears again in the command window showing the computer is ready for a new command. The appearance of the screen at this stage is shown in Figure 1.6.

- To find 86.29 – 43.31 type in

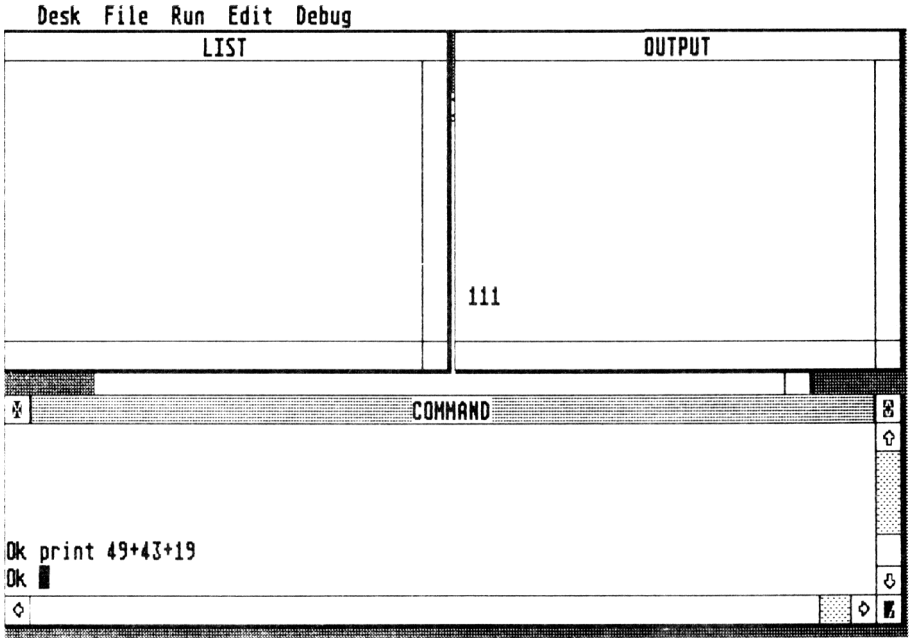


Figure 1.6 *Screen display after execution of the calculation command: print 49 + 43 + 19*

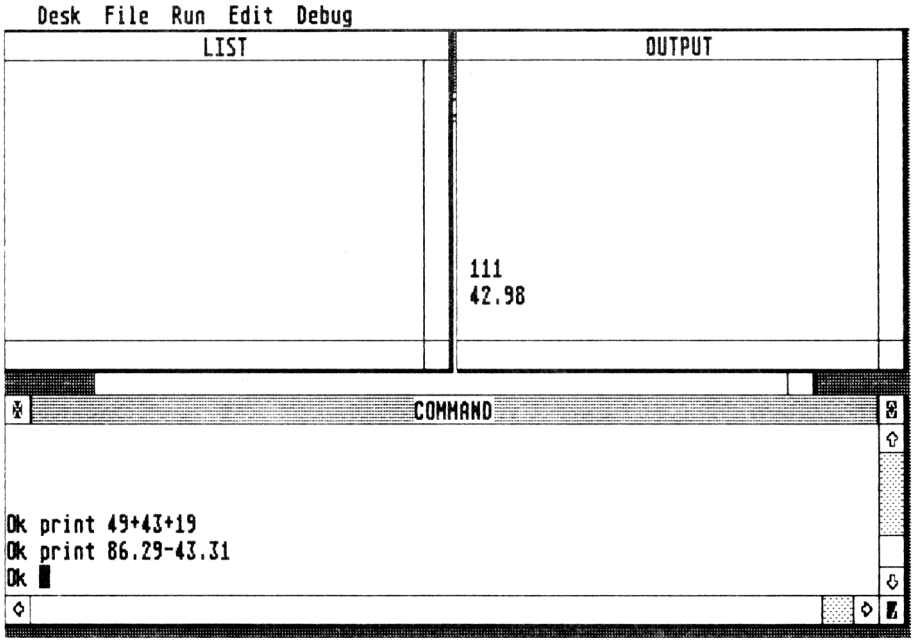


Figure 1.7 *Screen display after execution of second calculation command: print 86.29 − 43.31*

print 86.29−43.31 [RETURN]

The answer, 42.98, will be displayed in the output window and the screen display will be as shown in Figure 1.7. Note that either the full stop or the point on the number pad of the keyboard maybe used for the decimal point.

- To find 56.3 × 0.69, type in

  print 56.3*0.69 [ENTER]

  The answer, 38.847, is displayed in the output window with the previous two answers scrolling upwards to make room.

- To find 121 ÷ 42, type in

  print 121/42 [RETURN]

  The answer 2.88095, to six significant-figure accuracy, is displayed in the output window.

- To find 9.8 + 53.64 × 7.4 − 27.8 ÷ 0.4, type in

  print 9.8+53.64*7.4-27.8/0.4 [ENTER]

  The answer, 336.94, will be displayed in the output window.

If you have tried all the above calculations you will have almost filled the output window and already filled the command window as originally sized after loading the ST BASIC disk. Any further work is accommodated by the display scrolling automatically upwards.

Use of ? for print

We can use the ? key as a shorthand for the print command, e.g.

?47.3+42.9

and

PRINT 47.3+42.9

are identical. A further advantage in using ? for print is that you do not have to leave a space between ? and the calculation. In programs ? can also be used for print and will be listed as such in the program listing. One further point, you can use print in either lower or upper case, i.e. print and PRINT are identical. The computer makes no distinction between commands written in lower or upper case.

EXERCISE PROBLEMS 1.1
To gain some more practice try these:

1 456 + 876

2 75 × 67 × 32

3 452.78 ÷ 45.3

4 22.4 × 56.2 − 13.7 × 28.9

5 0.01672 ÷ 0.00034

Clearing the screen windows

Each of the four windows can be cleared by using the CLEARW N command:

CLEARW 0 [RETURN] clears the Edit window
CLEARW 1 [RETURN] clears the List window
CLEARW 2 [RETURN] clears the Output window
CLEARW 3 [RETURN] clears the Command window

Try now clearing both the output and command windows by entering:

CLEARW 2 [RETURN]

This command completely clears any display from the output window;

CLEARW 3 [RETURN]

clears the command window and returns the Ok caption plus cursor, i.e.

Ok■

to the bottom of the command window.

Making a window full size

We have explained previously that the size of a window can be easily changed by clicking on the particular window with the mouse pointer on that window so as to activate the window and then moving the pointer to the Size Box and dragging it until you achieve the size you want. You can also make a window full size by activating it first and then clicking on the Full Box.

ST BASIC also supplies a command for making a window full size, the

FULLW N command. N takes the same values as used in the CLEARW command, i.e. N = 0 for the Edit, N = 1 for the List, N = 2 for the Output and N = 3 for the Command window, e.g.

FULLW 2 [RETURN]

would make the output window full size. Both the CLEARW and FULLW commands are especially useful when used as statements within a program.

Further applications of PRINT

In this section we explain further uses of the print command including Print "...", the use of semi-colons and commas, the TAB function and PRINT USING.

The PRINT "..." command is used to instruct the computer to display characters, words, symbols, etc., typed within the double quotation marks. (Note: the characters enclosed within the quotation marks are known as strings in computer terminology.)

A semi-colon in a PRINT instruction acts as a separator which provides an immediate display with no interleaving spaces between successive string items or a single space between numeric items.

A comma in a PRINT instruction instructs the computer to print out successive items in zones consisting of 14 spaces. The first item after PRINT is allocated columns 1 to 14, the second item separated from the first by a comma is allocated columns 15 to 28, the third columns 29 to 42, and so on; if the character length of an item requires more than 14 then multiples of 14 columns are used.

The TAB function moves the cursor a specified number of spaces from the left-hand window margin, e.g.

PRINT TAB(n) "string"

will cause the string to be "printed" beginning at the nth column from the margin. The left-most position of the window corresponds to column 1.

PRINT USING allows us to specify a number or the result of a calculation in a given format, for example, to occupy a given number of spaces and contain a defined number of decimal places. The command takes the form

PRINT USING "###.##";x

where x is a number or calculation, and in this case means that the x value will

be formatted to have three places in front of the decimal point and two places behind. The hash symbol, #, is used to represent the digit position.

PRINT USING "##.####^^^^";x

specifies the value of x in exponential or standard form, e.g.

PRINT USING "##.### ^^^^"; 56.2*-892.3 [RETURN]

will produce the display −5.015E04 in the output window. [Note the E04 means 10^{-4}; E is used to denote power of 10 form.]

Examples

- The command,

  PRINT "Good morning" [RETURN]

  instructs the computer to display the words, i.e. the string, within the quotation marks in the output window. On pressing the RETURN key, the string is immediately displayed in the output window.

- Examples illustrating the action of semi-colons and commas used as separators:

  PRINT"x";"y";"z"[RETURN]

  produces the display in the output window of

  xyz

  i.e. no interleaving spaces, and

  PRINT "x", "y", "z" [RETURN]

  produces the display:

  x y z

  with x in column 1, y in column 15 and z in column 29.

- Next let us see how we can combine the display of both strings and the results of a calculation in a single PRINT command. For example, suppose we wish to display:

  74.2 × 53.8 = answer

  i.e. the calculation as a string as well as the actual answer. Type in:

print "74.2×53.8="; 74.2*53.8 [RETURN]

Here, "74.2×53.8=" instructs the computer to display the string within the quotation marks, 74.2*53.8 instructs the computer to work out and display the result of the calculation and typing in a ; means the answer will be displayed one space only to the right of the string.

- Examples of the use of TAB:

  print tab(5)"A" [RETURN]

  will result in A being displayed in the fifth column (four spaces and then A) from the left-hand margin of the output window.

  print tab(5)"A";tab(10)"B";tab(20)"C"[RETURN]

  produces the display:

  ```
      A     B              C
  ```

  i.e. A, B and C in 5th, 10th and 20th columns, respectively.

- Try typing in this short BASIC program. Although we have not covered this yet the program is quite easy to understand and illustrates the action of the tab function:

  ```
  10 for n=1 to 10 [RETURN]
  20 print tab(n)"*" [RETURN]
  30 next n [RETURN]
  ```

  When you have completed the three lines, run the program using the command,

  run [ENTER]

  The display obtained is shown in Figure 1.8.

- PRINT USING examples. The Atari ST series computers give results normally to six significant figures and for many purposes this is more than is required. For example,

  PRINT 46.2*3.67/7.2 [RETURN]

  displays the results as 23.5491. Suppose we wish the result to two decimal places, then use:

  PRINT USING "##.##"; 46.2*3.67/7.2 [RETURN]

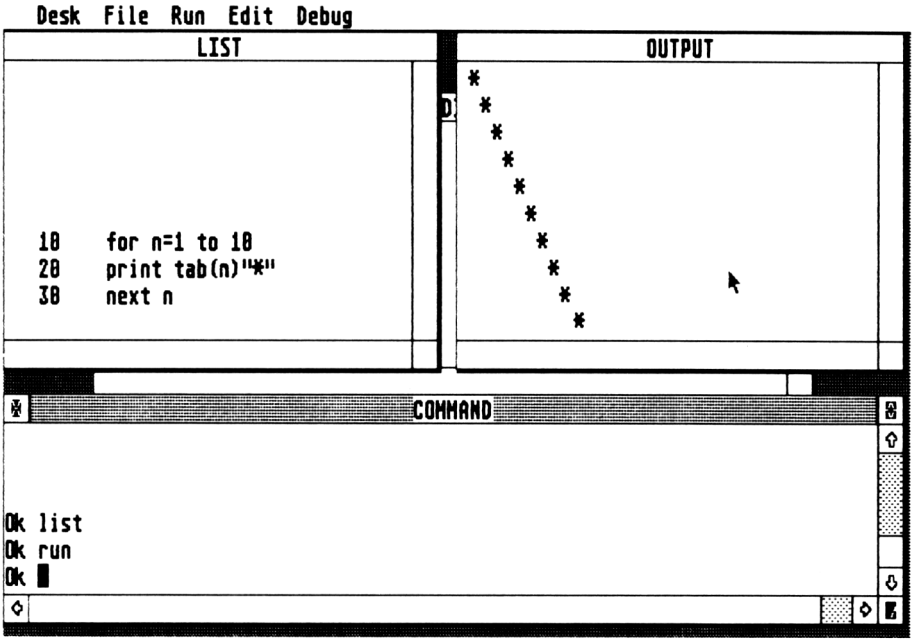


Figure 1.8 *Display obtained on running TAB illustration program*

23.55 will be displayed

For very large numbers and very small numbers standard form format is usually the most convenient, e.g.

PRINT USING "##.### ^ ^ ^ "; 2988*5624 [RETURN]

displays the result as 1.680E+08, i.e. 1.68×10^8.

PRINT USING "##.#### ^ ^ ^ "; 4.9/63781 [RETURN]

displays the result as 7.682E-05, i.e. 7.682×10^5.

EXERCISE PROBLEMS 1.2

1 Calculate 56.7 ÷ 3.3 and display in the output window:
(a) just the result
(b) the string "56.7/3.3=" as well as the result
(c) the result to two decimal places.

2 Calculate the following using a single print statement:

4.62×3.8, $57.33 \div 6.3$, $42378.6 - 33499.7$

3 Try to display your name followed by your address.

Drawing lines, circles and ellipses

ST BASIC provides the following graphics type commands for drawing lines, circles and ellipses:

LINEF x1,y1, x2,y2 draws a line in the output window connecting points (x1,y1) (x2,y2).

CIRCLE x,y,r draws a circle, centre (x,y) and radius r.

ELLIPSE x,y,a,b draws an ellipse, centre (x,y) with a horizontal (x-axis) half-diameter value of a and a vertical (y-axis) half-diameter value of b.

All units for the coordinates, points and lengths used in these commands are in pixels (picture element units). For a full-size output window there are approximately 600 pixels horizontally across the window and 400 pixels in the vertical dimension. The origin (0,0) point of the output window corresponds to the top left-hand corner, the x-axis runs horizontally and the y-axis vertically downwards as shown in Figure 1.9.

Try these commands:

LINEF 0,0, 200, 150 [RETURN]

produces a straight line between point (0,0) and (200,150) as shown in Figure 1.10(a).

CIRCLE 100, 100, 60 [RETURN]

produces a circle, centre at (100,100) and of radius 60 as shown in Figure 1.10(b). Note unless you have previously cleared the output window using the CLEARW 2 command, the line produced by the LINEF command above will also be present.

ELLIPSE 120, 100, 90, 45 [RETURN]

draws an ellipse as shown in Figure 1.10(c).

Arcs of circles and ellipses can also be drawn by adding two further values – the start and stop angles of the arc – to the parameter list following the

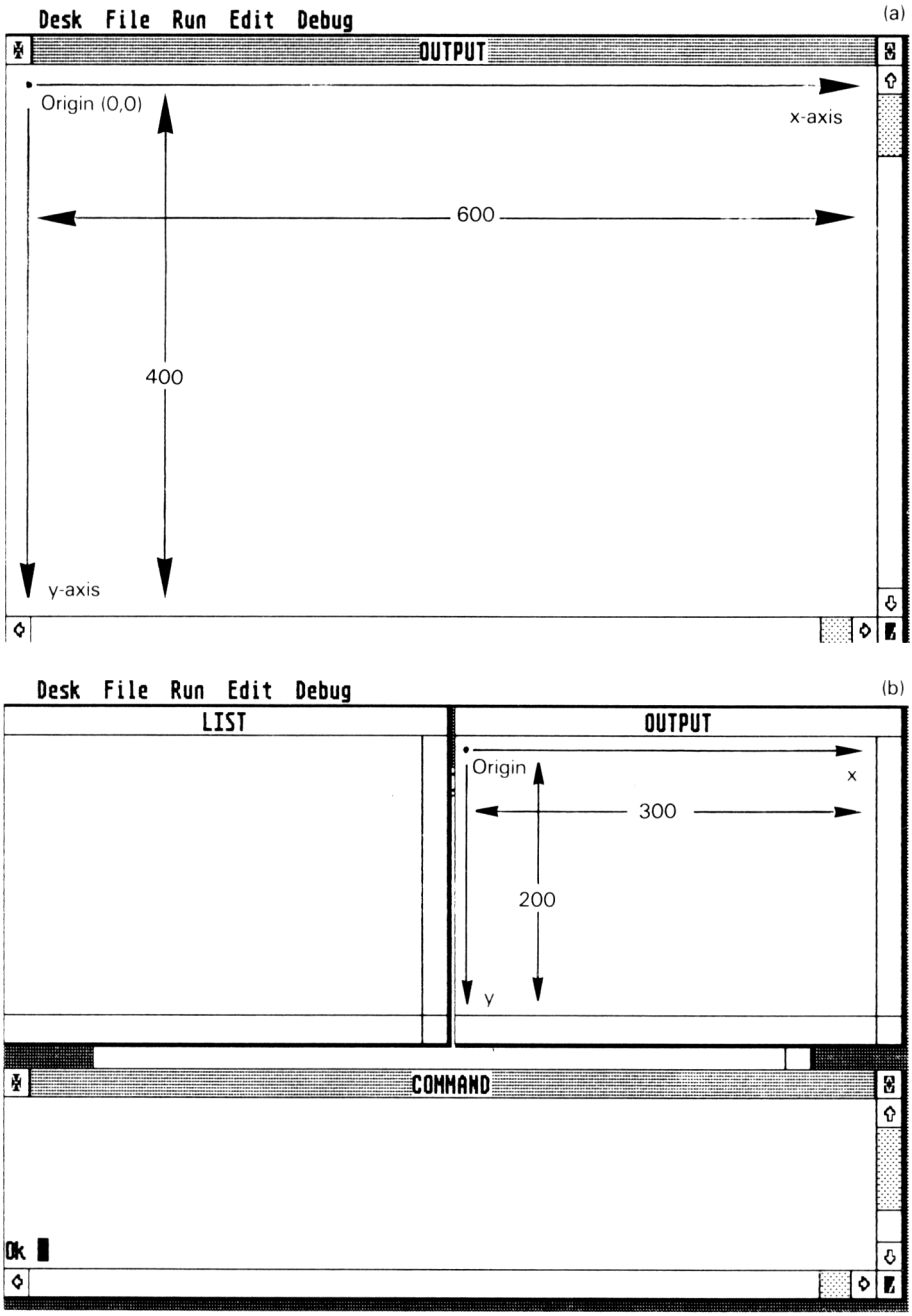
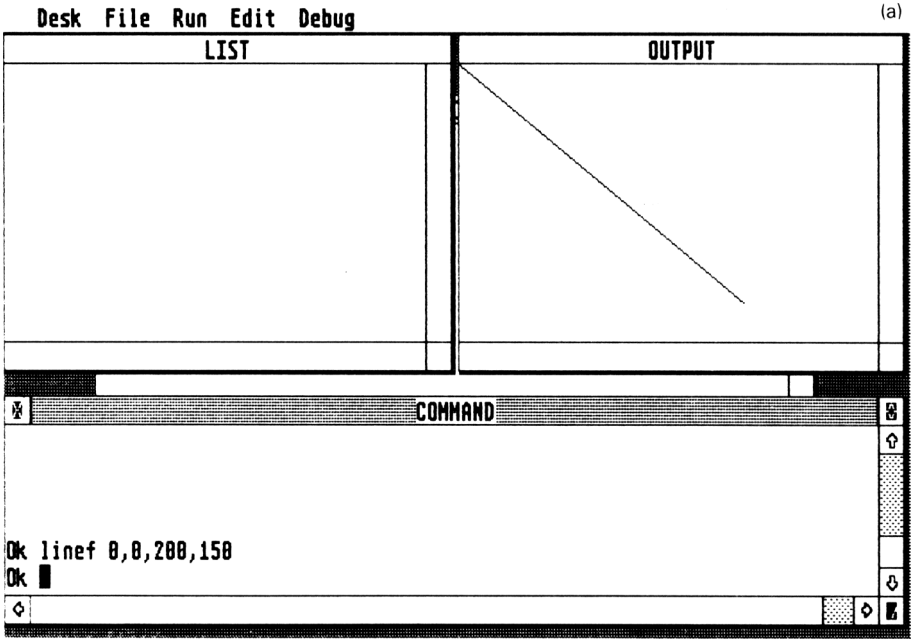
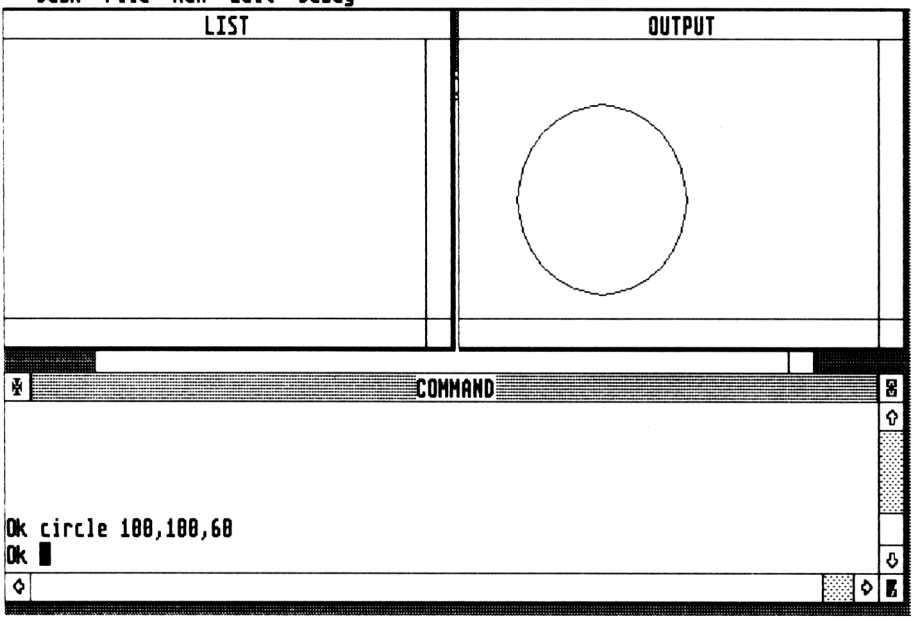


Figure 1.9 *Coordinate system for graphics commands in output window. Origin is at top left-hand corner, x axis runs horizontally, y axis runs vertically downwards.* (a) *When output window is made full size;* (b) *situation immediately after loading ST BASIC disk*

(a)


Desk File Run Edit Debug
LIST
OUTPUT
COMMAND
Ok linef 0,0,200,150
Ok

(b)


Desk File Run Edit Debug
LIST
OUTPUT
COMMAND
Ok circle 100,100,60
Ok

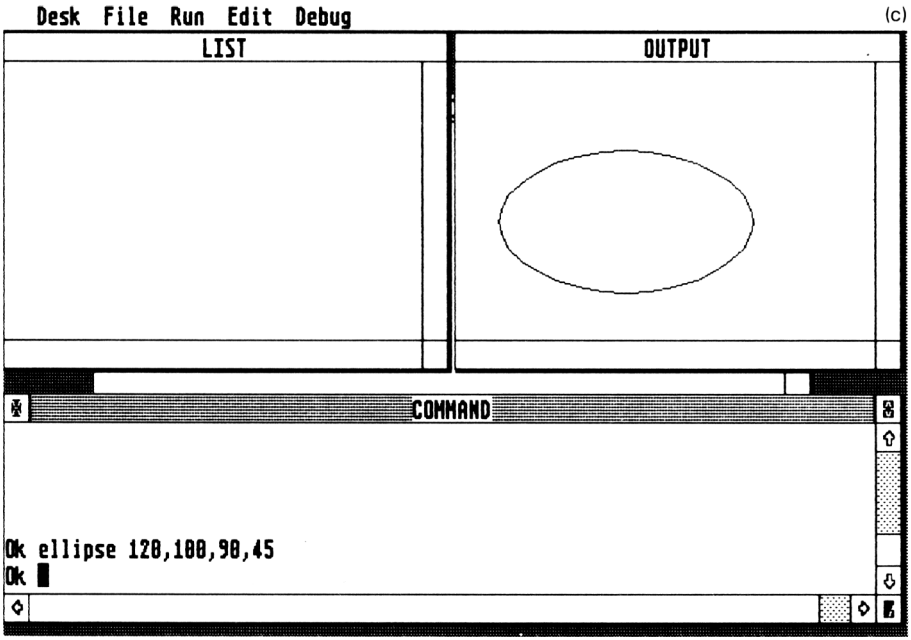


Figure 1.10 *Examples of output display produced by graphics commands:* (a) *LINEF 0,0,200,150;* (b) *CIRCLE 100,100,60;* (c) *ELLIPSE 120,100,90,45*

command list. Note that both these angles are entered as degrees × 10, e.g. for start and stop angles of 90 degrees and 180 degrees enter 900 and 1800, respectively.

Thus, for example, the command

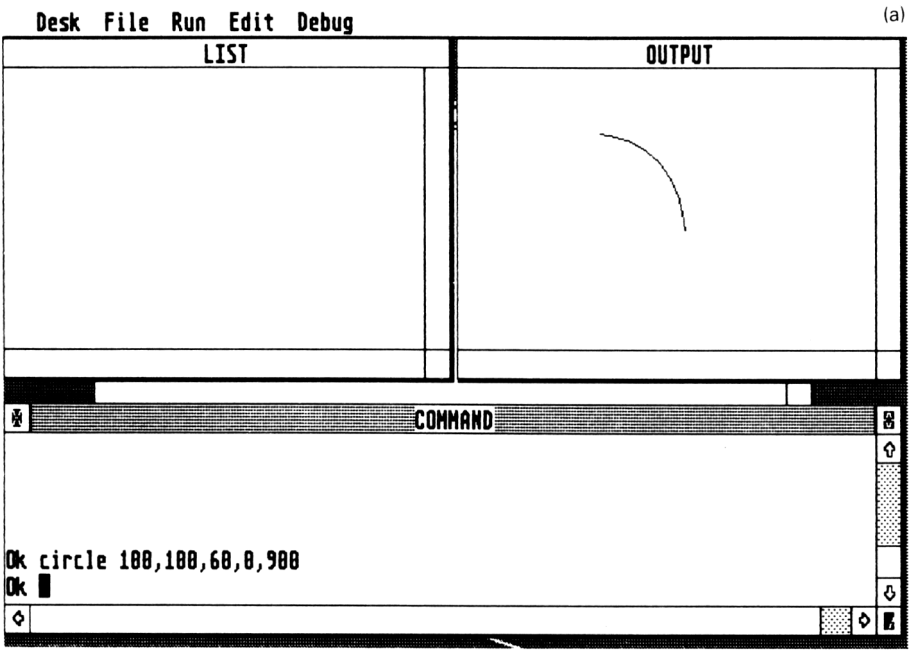
CIRCLE 100, 100, 60, 0, 900 [RETURN]

draws the arc of the circle, centre (100,100) and radius between 0 degrees and 90 degrees, i.e. in the first quadrant as shown in Figure 1.11(a); whilst the command

ELLIPSE 120, 100, 80, 40, 900, 2700 [RETURN]

draws the arc of an ellipse between 90 degrees and 270 degrees, in the second and third quadrants as shown in Figure 1.11(b).

Solid, i.e. coloured in, circles and ellipses can be drawn using the PCIRCLE and PELLIPSE commands in conjunction with the COLOR command. The PCIRCLE and PELLIPSE commands have identical parameter list formats as



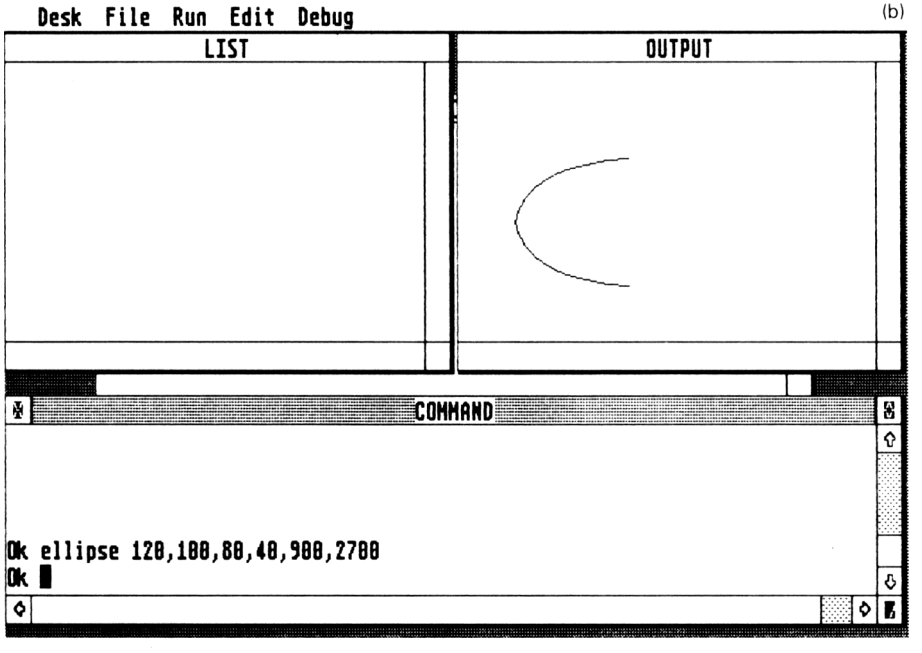


Figure 1.11 *Drawing arc lengths: arcs produced by* (a) *CIRCLE 100,100,60,0,900;* (b) *ELLIPSE 120,100,80,40,900,2700*

the CIRCLE and ELLIPSE commands but the outlines they produce will be filled in by the fill colour set up by the COLOR command.

This command may be used to set text, fill and plot colours and takes the form,

COLOR text colour no, fill colour no, plot colour no.

and for black and white colours only we need only remember

colour number for black = 1
colour number for white = 0

Thus if we wish to draw a solid, black-filled circle, we must first set the fill colour to black using the command

COLOR 1,1,1 [RETURN]

which in facts sets all three text, fill and plot colours to black. Then type in,

PCIRCLE 100,100,60 [RETURN]

which will draw a solid black circle centre (100,100) of radius 60 as shown in Figure 1.12(a).

Now if you change the plot colour to white by the command,

COLOR 1,1,0 [RETURN]

and also type in

CIRCLE 100,100,30 [RETURN]

you will now observe the white circle outline of radius 30 and same centre as before within the bigger black solid circle as shown in Figure 1.12(b).

ST BASIC also provides a FILL command. By using this command in conjunction with COLOR and the graphic plot functions we can draw outlines and then fill in particular bounded areas with a given colour. The FILL command has the form,

FILL x,y

where x,y define the coordinates of a point within the bounded region we wish to fill in.

The use of FILL is illustrated in the following program. We use a program this time to tie our command together in a logical sequence of steps. Each line of

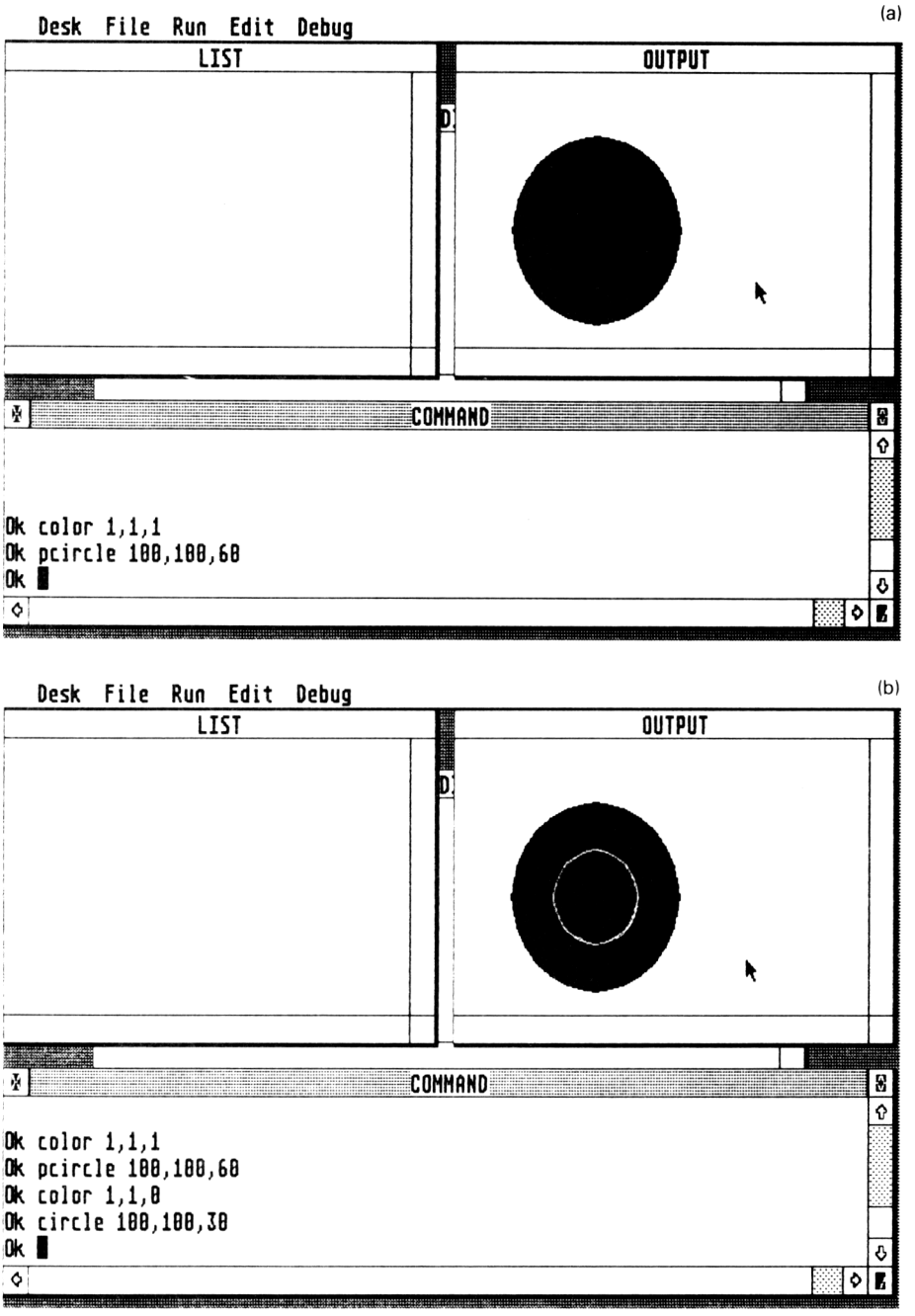


Figure 1.12 (a) *PCIRCLE and* (b) *COLOR commands*

the program is in fact a command and will be executed in ascending order when the program is run:

```
10      clearw 2
20      color 1,0,1
30      circle 100,100,70
40      circle 100,100,50
50      circle 100,100,30
60      circle 100,100,10
70      color 1,1,1
80      fill 60,100
90      fill 100,100
```

Type in each program line preceded by its number and terminated by [RETURN]. When complete type in the RUN command to execute the program, i.e.

RUN [RETURN]

You will obtain the display shown in Figure 1.13.

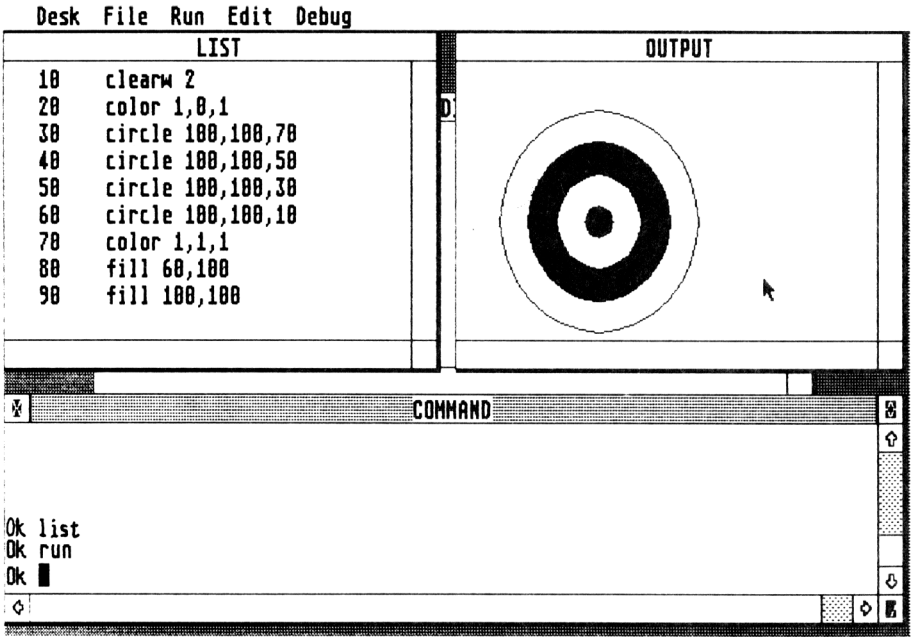


Figure 1.13 *Screen display produced by program illustrating action of FILL function*

The explanation of the program is as follows:

Line 10 clears the output window
Line 20 sets the text colour to black, the fill colour to white, the plot colour to black
Line 30 draws circle outline centred at (100,100) and radius = 70
Line 40 draws circle same centre, radius = 50
Line 50 draws circle same centre, radius = 30
Line 60 draws circle same centre, radius = 10
Line 70 this command now changes the fill colour to black
Line 80 the coordinates (60,100) place the "fill" point inside the region bounded by the circle of radius 50 and the circle of radius 30; this region will therefore be filled with the black colour
Line 90 the fill point (100,100) is bounded by the smallest circle of radius 10 and it is therefore this region that is filled in black.

Further calculations

Here we consider some further calculations involving the use of the ^ key for finding powers (^ is the upper case character on the 6 key), SQR(X) for finding square roots and the brackets, (), keys for more involved problems.

To find powers: use of ^ key

To find 2^{15}, type in the command:

print 2 ^ 15 [ENTER] or [RETURN]

The answer, 32768, is displayed in the output window.

To find 1.085^{20}, type in

print 1.085^20 [ENTER]

The answer, 5.11203, is displayed in the output window.

To find $5.4^{1.4}$, type in

print 5.4^1.4 [ENTER]

and 10.6011 is displayed in the output window.

To find square roots: use of SQR(X)

ST BASIC supports a number of standard functions, which we consider in some detail in Chapter 5. SQR(X) is one of them. It computes the square root of the number of numeric expression X enclosed by the brackets.

To find $\sqrt{2079.36}$, type in

print SQR(2079.36) [ENTER] or [RETURN]

The answer, 45.6, is displayed in the output window. Note the () brackets – upper case on keys 8 and 9 of main keyboard – must be used; or more easily the brackets in association with the number keys on the right-hand side of the keyboard.

We could, of course, use the ^ key, i.e.

print 2079.36 ^ 0.5 [ENTER]

The answer, 45.6, is displayed in the output window.

To find $\sqrt{0.00978}$, type in

print SQR(0.00987) [ENTER]

The answer displayed is 9.93478E-02. Note 9.935E-02 is equivalent to $9.935 \times 10^{-2} = 0.09935$.

Using brackets

When working out calculations involving several different operations, e.g. addition, and/or multiplication, division, finding powers, etc., we must instruct the computer with the correct order. The computer acts according to the usual rules of precedence:

1 It evaluates terms contained in brackets first.

2 It carries out multiplication and division (from left to right) before addition and subtraction.

Thus, to evaluate

$$\frac{5.6 \times 2.3}{10.2 - 9.4 \times 0.7}$$

we must enter the expression into the computer using brackets to group the numerator terms together and likewise the denominator, i.e. type in

print (5.6*2.3)/(10.2−9.4*0.7) [ENTER] or [RETURN]

3.5580111 will be displayed in the output window.

To evaluate

$$\frac{61.4 - 29.5}{4.8(56.2 + 33.6)}$$

type in

print (61.4-29.5)/(4.8*(56.2+33.6)) [ENTER]

The * must be inserted as must the outer brackets to hold together the total denominator. The answer displayed is 7.400705E−2.

To evaluate $[21.7/(3.8 + 6.5/3.5)]^5$ to two decimal places, type in

print using "###>##"; (21.7/(3.8+6.5/3.5))^5 [ENTER]

830.45 is displayed.

EXERCISE PROBLEMS 1.3
Try working out these:

1 $\sqrt{(5.62 \times 3.98)}$

2 $1500\,(12.6 + 9.4)^2$

3 39.6^2

4 $\dfrac{5.6(41.6 + 33.4)}{144.5 - 96.2}$

5 $\left(\dfrac{4.7 + 3.6 \times 2.9}{36.2 + 45.8}\right)^3$

CHAPTER 2

BEGINNING TO WRITE AND RUN PROGRAMS

Introduction and summary

In this chapter we begin to write simple but complete programs on the Atari ST series computer. We introduce the chapter with a brief discussion of what a program is, the form an ST BASIC program takes and some important points to be noted in program construction.

We then consider some of the important ST BASIC commands and instructions that will be used to form our program statements, to execute the program and to display a listing of the program contents. Also considered are the meaning of variables and allowable identifiers that may be used within programs for handling data; how a program may be edited; how a program may be saved on disk and how a program stored on a disk may be loaded and "run"; how parts of programs can be merged; how files can be deleted and finally how programs can be systematically traced through line-by-line for error checking.

What is a program?

So far we have, with a few exceptions been entering direct commands into the computer. These commands have been directly executed by the computer with immediate action. Clearly we are very limited in using only direct commands to solve our problems or execute all but the simplest of tasks. The idea of a program is to tie our set of commands together in a complete self-contained package which we can "run" at one go.

WRITING AND RUNNING PROGRAMS

What is a program? A program consists of a complete list of consecutive instructions designed to solve a particular problem or execute a given number of tasks.

To provide an immediate idea of the form taken by a simple ST BASIC program, consider the following example:

```
10    clearw 2
20    fullw 2
30    print "Capital Cities"
40    print "--------------"
50    print "England.............London"
60    print "France..............Paris"
70    print "Denmark.............Copenhagen"
80    print "Italy...............Rome"
90    print "Holland.............The Hague"
100   print "Brazil..............Brasilia"
110   print "Pakistan............Islamabad"
120   print "Australia...........Canberra"
130   print "Poland..............Warsaw"
140   print "Spain...............Madrid"
150   print "China...............Peking"
```

This simple program consists of a list of 15 instructions. Each instruction is typed in as a separate line statement. A line statement is an instruction to the computer to perform a specific operation.

Each line statement must always be preceded by a line number, the value of the numbers dictating the order in which the program will be executed, i.e. line statement 10 will be executed first, followed by 20, 30, 40,...,150 in the above example. Any ascending-order sequence can be used for the line numbers. We chose 10, 20,...,150, but we could have used 1, 2, 3, 4, 5,...,etc., or 15, 26, 48, 115, 207..., in fact any sequence in the range 1 to 65535; 65535 is the maximum line number for the Atari ST computers.

In typing in a program from the keyboard, each line statement is entered into the computer by pressing either the RETURN key or the ENTER key. The action of RETURN and ENTER is to store the line in the computer's memory. No further action is taken until the program is commanded to RUN. This is different from using the computer in the direct mode, where, for example

PRINT "Rome is the capital of Italy" [RETURN]

would produce immediate (direct) execution and cause

Rome is the capital of Italy

to be displayed in the output window.

Thus if a line is not preceded by a line number then the line statement is interpreted as a direct command and the computer processes the statement immediately. When the statement is preceded by a number then the statement becomes part of a program.

Now try typing in the program. Remember to press the RETURN key at the end of each line. The program can be run by typing in the direct command

RUN [RETURN]

Alternatively, use the mouse and its pointer to select the Run menu. This is located on the menu bar at the top of the screen. The Run menu will drop down; position the pointer on the Run option and just click on the left-hand mouse button. Your program will run just in the same way as if you had typed in RUN [RETURN].

RUN causes execution of lines 10, 20, 30, 40, 50...150 in ascending order and produces immediately the following display in the output window:

```
Capital Cities
--------------
England.............London
France..............Paris
Denmark.............Copenhagen
Italy...............Rome
Holland.............The Hague
Brazil..............Brasilia
Pakistan............Islamabad
Australia...........Canberra
Poland..............Warsaw
Spain...............Madrid
China...............Peking
```

Before proceeding further, let us broadly review some basic points to be considered in writing a program:

- **Ensure that you have an exact understanding of what you want your program to do. Define your problem as clearly as possible.**

  Perhaps obvious, but absolutely essential as your first step for any serious problem-solving using a computer. Note what information you are given –

this will constitute the data input to your program (we see how to input information using INPUT and READ-DATA statements in the next chapter).

Define clearly what information output you require from your program.

- **Develop general ideas for your program solution. Think out a suitable sequence for your instructions.**

  Draft a logical sequence of the actions required to solve your problem. From these form the basis of what type of line statements or, as we shall see in Chapter 4, what subroutines are needed to effect your solution.

- **Type in the program line statements.**

  Remember that each statement must be preceded by a number and that the computer acts on these statements in ascending number order. It is a useful tip to separate consecutive lines by 10, e.g. use 10, 20, 30,. . ., etc., so that if you subsequently need to insert lines in your program you have plenty of space to do so.

  All line statements must, of course, be written according to the syntax (the "grammar" rules) dictated by the version of ST BASIC. If these are not followed exactly the program will not run. In practice, the program is continuously monitored as each line is typed in. After each line is entered, the computer scans it for errors and will display an error message, "something is wrong", if any should be present, and give guidance as to the type of error detected.

- **Feed in any data (when and where necessary) to the program.** This may be done interactively using INPUT statements or by storing data within the program itself using READ-DATA statements.

Fundamental commands and statements

Let us now review some of the important ST BASIC commands and statements that we will be using to run and write our programs.

Fundamental direct mode commands

RUN This command tells the computer to execute or 'RUN' our program; the processing starts, in the absence of any number typed in, after RUN at the lowest number line statement in our program.

Although RUN can be used within a program, its normal use is to start program execution by typing RUN as a direct mode command. Using the Run

menu option and clicking on RUN does the identical job.

RUN 40 would start execution at line statement number 40.

LIST This command displays the program currently stored in the computer in the list window.

Note: the LIST command can be brought into operation by selecting the Edit menu on the Menu bar. This will drop down. Select LIST, click on it and the program listing will be displayed in the list window.

LIST 10-100 will list the group of program statements starting at 10 and ending at 100.

LIST 50- will list from line 50 to the end of the program.

LIST -70 will list from start to line 70.

LIST 90 will list specified line 90.

NEW This command clears all programs stored in the computer – it allows us to start afresh or "a-new"

AUTO Use of the AUTO command automatically assigns for us line numbers in steps of 10, i.e. 10,20,30,40, during the writing of our program; to cancel the AUTO function, press and hold down the CONTROL key and press the G key.

RENUM This is used for renumbering the program statements, e.g. at 10 with increments of 10, i.e. line numbers would be renumbered 10, 20, 30,. . . .

RENUM 100, 10, 200 renumbers program lines from "old" line number 10, to give it a new number 100 and increments subsequent lines by 200; the general form of the command is:

RENUM new line number, old line number, line increment

Note: to use the RENUM function you must have a disk in the drive unit which is not write-protected; this is because RENUM actually creates a file (called BASIC.WRK) to do its job.

DELETE This command deletes a line, e.g.

DELETE 50 deletes line 50.

DELETE -100 deletes all lines up to and including line 100.

DELETE 200- deletes all lines from 200 to end of program.

DELETE 20-80 deletes range of lines.

CLEARW N clears the BASIC windows. When

N=0 the Edit window is cleared
N=1 the List window is cleared
N=2 the Output window is cleared
N=3 the Command window is cleared

BREAK This command stops program execution after every line and is affected by selecting the Run option on the menu bar and then clicking on Break. Both the program line and any output are displayed. Pressing RETURN will cause the next line to execute. BREAK is cancelled by the UNBREAK command.

Some fundamental program statements

Assignment statements: the LET statement

```
100 LET X=98.2     or equivalently
100 X=98.2
```

This statement assigns to the variable X the value 98.2. The use of the keyword LET is optional in ST BASIC and therefore may be omitted. Thus the two assignment statements given above are identical. In this book LET will always be omitted in writing assignment statements. The use of the = sign in a LET statement does not have quite the same significance of equality as in normal algebra or arithmetic. For example, the program

```
100 SUM = 527
110 X = 243
120 SUM = SUM + X
130 PRINT SUM
RUN [RETURN]
770 . . .result obtained
```

i.e. the result yielded on running the program is SUM = 527 + 243 = 770. Statement 120 is a nonsense in normal algebraic terms (unless X = 0). In computing, however, the assignment statement = sign means "LET left-hand

variable take the value of the right-hand expression" or more briefly "is given the value of." Thus statement 120 is read as "Let the variable SUM take the value of the original SUM value, i.e. 527 plus the value of X, i.e. 243".

Output statements – PRINT statements We have already used the PRINT command in direct mode usage. PRINT is used in an identical way in program statements. For example,

```
120 PRINT X
```

instructs the computer to display the value currently assigned to X in the output window.

```
140 PRINT "Solution ="; X
```

displays the words within the quotation marks (the string) immediately followed by the value of X.

```
160 PRINT
```

"prints" a blank line and is very useful in separating results, etc., for a clear output display.

The REMark statement This statement is used to include a REMark or comment, e.g.

```
10 REM ** cash-flow program **
```

It can also be used to put in program title or any helpful explanatory note; it is for our convenience only and is passed over during the execution of the program. The ' (apostrophe) can be used as an abbreviation for REM in REMark statements.

Use of the colon More than one statement may be written on a single line by separating the individual statements by a colon. For example, the program

```
10 X = 242
20 Y = 251
30 Z = 363
40 PRINT X + Y + Z
```

could be condensed to

```
10 X = 242: Y = 251: Z = 363
20 PRINT X+Y+Z
```

or even into a single line,

10 X = 242: Y = 251: Z = 363: PRINT X+Y+Z

Use of CLEARW N as a program line statement This statement is used to clear the windows, for example,

10 CLEARW 2

will clear the display in the output window and it is helpful to include such a statement at the beginning of a program to ensure an initially clear output window.

20 CLEARW 0 : CLEARW 1 : CLEARW 3

i.e. N = 0, 1, 3, would respectively clear the edit, list and command windows.

Writing and running simple programs

This section gives four examples of very elementary programs that show the use of some of the commands discussed above.

Names and phone numbers

Try writing a program to display the names and phone numbers of your friends. Here is the form the program could taken:

```
10      rem *** Phone numbers of friends ***
20      clearw 2
30      print "John" tab(15) "65 89 310"
40      print "Hazel" tab(15) "112 437 90"
50      print "Mike" tab(15) "14 652 009"
60      print "The George" tab(15) "007 78231"
70      print "St. Marys" tab(15) "4435 6709"
```

Remember each line statement has its own line number and must always be "entered" by pressing the RETURN key. Try using AUTO for automatic line numbering.

To execute the program you can either type in RUN followed by pressing the RETURN key, or select the Run menu and click on the Run option.

You will of course, obtain the following display in the output window

```
John            65 89 310
Hazel           112 437 90
Mike            14 652 009
The George      007 78231
St. Marys       4435 6709
```

The program is easy to understand. Line statement 10 contains REM (RE-Mark), a remark statement to "remind" us what our program does. Statement 20 used CLEARW 2 to clear the output window. Statements 30 to 70 instruct the computer to "print" the names and phone numbers; we have also used the TAB instruction to line up the start of the numbers, i.e. TAB (15) causes the numbers to be displayed commencing at column 15 of the output window.

If you now type in LIST [RETURN] you will obtain a display of the program listing in the list window. If you follow this by NEW [RETURN] you will clear the program from the computer memory, although not the display on the screen.

Check this by typing in,

CLEARW 1 [RETURN]
LIST [RETURN]

The CLEARW 1 command clears the list window. The LIST command lists the program currently stored in the computer. Since no "list" appears, the program was in fact cleared by the NEW command.

Note the BASIC keywords run, list, print, clearw 1...etc., can be entered in in either lower or upper case letters. The computer makes no distinction.

Displaying initials

Try displaying your initials, either in direct or some contrived form using PRINT statements. Here is an example, displaying mine:

```
10    rem *** Initials ***
20    clearw 2:fullw 2
30    print tab(20)" *****  *****   *      * "
40    print tab(20)" *    *  *    *  **    ** "
50    print tab(20)" *    *  *       * * * * "
60    print tab(20)" *****  *  *** *  *  *  * "
70    print tab(20)" * *    *    *  *     * "
80    print tab(20)" *  *   *    *  *     * "
90    print tab(20)" *   *  *****  *     * "
```

To direct the output to a printer use LPRINT, e.g.

100 LPRINT "Result = " X

would direct the output to be displayed on a printer provided, of course, a printer is correctly connected to the printer port at the back of the computer.

Area of a triangle

Here is an example of a simple program to calculate the area of a triangle (see Figure 2.1):

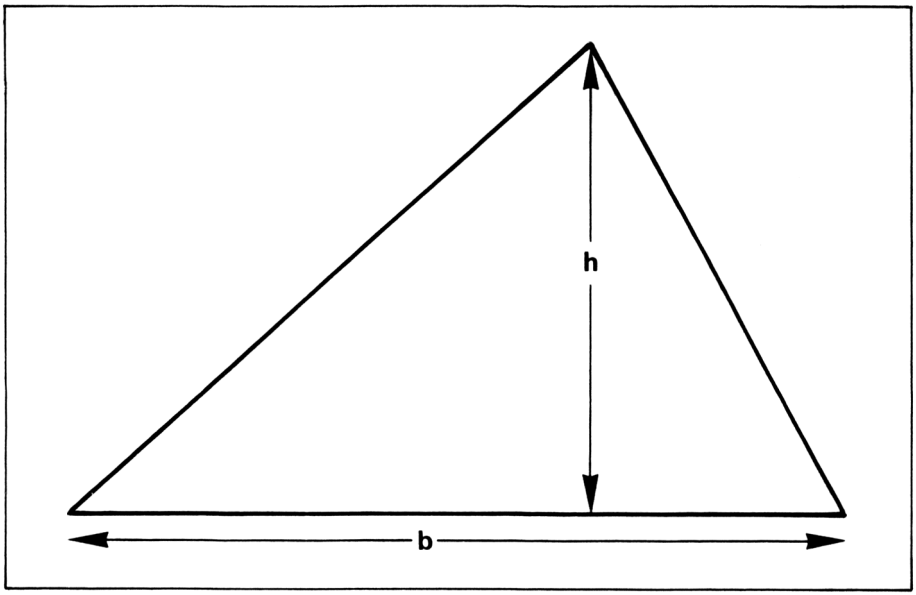


Figure 2.1 *Area of triangle = ½ × base × height (A = ½bh)*

```
10    rem *** Simple area calculation ***
20    clearw 2
30    lprint "Area of triangle =";
40    lprint "base x height / 2 "
50    lprint
60    tbase = 38.4
70    theight = 23.6
80    tarea = tbase*theight/2
90    lprint "  Area of triangle = ";tarea
100   lprint "  ------------------------------"
110   lprint "            for base = ";tbase
120   lprint "          for height = ";theight
```

Again the program is reasonably easy to understand. Line 40 assigns the base 38.4, i.e. tbase = 38.4. Line 50 assigns the height as 23.6. Line 60 assigns the area the value half base times height and line 70 displays the result in the output window. The full display obtained is

```
Area of triangle =base x height / 2

  Area of triangle =   453.12
  ---------------------------
           for base =   38.4
         for height =   23.6
```

Cylinder in perspective

This program draws a cylinder in perspective. The ELLIPSE and LINEF statements are used to accomplish the task:

```
10      rem *** Drawing a cylinder ***
20      clearw 2: fullw 2
30      ellipse 200,50,100,40
40      ellipse 200,300,100,40
50      linef 100,50,100,300
60      linef 300,50,300,300
```

The display obtained in the output window is shown in Figure 2.2.

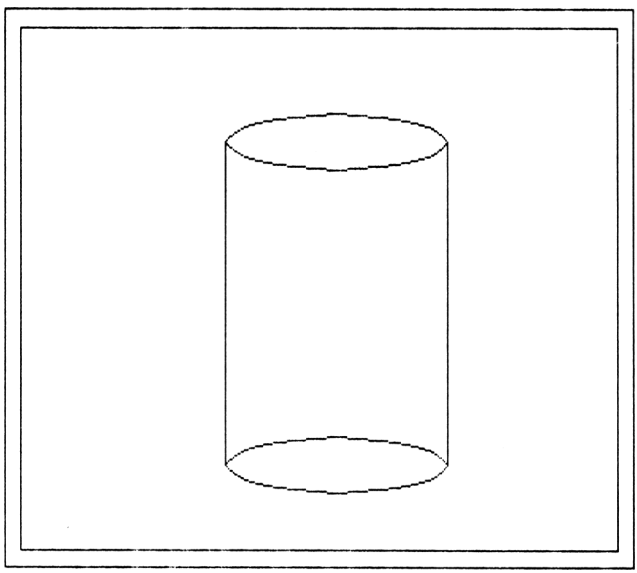


Figure 2.2 *Output window display for cylinder program*

Variables and identifiers

When we assign values to quantities to be used in our programs we need a "box" or storage location to hold these values. In computer terminology we talk about variables (the word used for the quantities we wish to process, etc., in our programs) and identifiers, the name we "identify" or label our "variable" box.

The term variable refers to any element or quantity in a program whose value may, if desired, be changed in the execution of the program. Variables are used to provide storage locations in the computer's memory to hold the data the variables represent.

Each variable we use in a program must be given a unique identifier. The variable identifier may be thought as the name of the "container" used to store the data value currently assigned to the variable. Once a value is assigned to a given variable, the value remains fixed at this value and stored in its "container", but may be changed when or if any subsequent program statement assigns the variable a new value.

In ST BASIC there are three main classes of variable:

Real or numeric variables

These are used to store numbers (with or without decimal parts) in floating-point form within the computer, to six-significant-figure accuracy in the case of the Atari ST computers.

Identifiers for all variables must always begin with a letter of the alphabet and then can be followed by any sequence of letters and numerical digits (and for the Atari computer can include the full stop symbol). Spaces, however, are not allowed. The final character for a real variable is the ! (exclamation mark). This symbol is optional and may be omitted but is sometimes useful to include as it distinguishes real variables from integer and string variables. The total character length for an identifier must not exceed 31.

Obviously no BASIC keywords, e.g. print, list, or renum, can be used as variable identifiers. All keywords, system commands, etc., are regarded as reserved words and must never be used except in their correct context. Note also that spaces, punctuation marks (other than the full stop), and arithmetic symbols cannot be used in identifiers. For example

a!	maxvalue	Z237	GREATEST
area!	min.point	interest	least
x11!	last.value	POPULAR	no.of.revs

are all valid identifiers for real (number) variables; the following, however, are not:

max value!	invalid, spaces cannot be used in identifiers
x+y	invalid, arithmetic symbols cannot be used
input	invalid, BASIC reserved word
first.value?	invalid, ? cannot be used

Integer variables

These are used to store whole number values, i.e. integers. The range for the Atari ST computers is −32768 to +32767. Integer variable values occupy less space in the computer memory and so can be used in preference to real variables in programs that do not require decimal information.

Integer variables are distinguished from real variables by including the % symbol as the last character of the identifier. Integer variable identifiers follow exactly the same syntax rules as for real variables, but must always end with %, e.g.

index% represents an integer variable identifier
index a real (floating-point) variable identifier

String variables (non-numeric variables)

These as their name implies are used to store "strings" of characters, i.e. letters, numbers, symbols, etc. String variable identifiers are distinguished from numeric variables by always ending in the $ (dollar) symbol. It is the $ symbol that "tells" the computer that it is handling string variable data. Apart from the $ symbol, string variable identifiers are formed using the same general rules: they must start with a letter, may then be followed by any sequence of letters, numbers or full stops (up to a maximum including the $ symbol of 31), e.g.

surname$ fname.and.adress$
x11$ SUBJECT.3$

are valid string variable identifiers.

We use assignment statements to assign values to string variables in the same way as for numeric variables, but we must enclose the actual value being assigned (the string) within double quotation marks, as shown for example in the following short program:

```
10    rem *** String variable example ***
20    clearw 2
30    firstname$ = "Richard"
40    blank$ = " "
50    address$ = "42 Peshawar Avenue, N.W.F.P."
60    print firstname$
```

```
70      print address$
80      print firstname$;blank$;address$
```

On running the program, we obtain the display

```
Richard
42 Peshawar Avenue, N.W.F.P.

Richard 42 Peshawar Avenue, N.W.F.P.
```

String variables can be combined to form larger strings using the + symbol. For example, in the following program at line 60:

total$ = n$ + a$

assigns the string variable total$ the value:

"Richard 42 Peshawar Avenue, N.W.F.P.

Try running the program to verify this.

```
10      rem *** Second string variable example ***
20      clearw 2
30      n$ = "Richard"
40      b$ = " "
50      a$ =  "42 Peshawar Avenue, N.W.F.P."
60      line$ = "----------------------------------------"
70      total$ =n$+b$+a$
80      print total$
90      print line$
```

Upper- and lower-case usage

Identifiers written in either upper- or lower-case letters are treated as identical in BASIC. Thus, for example, the following pairs of identifiers are treated by the computer as the same:

X,x; N%,n%; firstname$, FIRSTNAME$

Do not fall into the trap of thinking they can be used for two different variables, they are indistinguishable as far as program execution is concerned.

Identifiers are used not only to identify variables, but, as we soon see for arrays, user-defined functions, loops, file-names, etc.

One final point – always try to choose meaningful identifiers for the variables, etc., used in your programs. This adds extra clarity to the program listing. It helps in understanding what your program is about, is especially useful in tracing errors and for future reference, and invaluable for others who may use your programs.

Obtaining hard copy

If you are lucky enough to have a printer – the Atari ST may be used directly with a parallel printer with Centronics interface, the Epsom MX, RX and FX series of printers are ideal – you can easily obtain a hard copy print out of your program listings and also direct your program output to the printer as well as or instead of the monitor screen.

The listing command is simply,

LLIST [RETURN]

If you wish your program output directed to the printer and printed then use LPRINT. This directs output to the printer in the same way as PRINT directs output to the output window, e.g.

```
100 LPRINT "Print results follow"
110 LPRINT
120 LPRINT X, Y, Z
```

Editing programs

The Atari ST computers have excellent editing facilities for making corrections, inserting lines, deletions, etc., and also for saving or loading the new text after alterations have been made. Mistakes in any given line of a program can, of course, be rectified whilst in the command mode by typing in the line again, removing the errors and pressing RETURN. Additional lines can also be added using appropriate line numbers in the same way. However, for long programs, command mode corrections are laborious and as the editor is so easy to use, almost self-explanatory, use it.

To enter the EDIT mode simply select the Edit menu on the top line of the screen and click on it with the mouse pointer. The Edit menu will immediately drop down as shown in Figure 2.3. Click on the Start Edit option, the menu will disappear and be replaced by the Edit window containing your program lines (or for a long program, the initial part), as shown in Figure 2.4. Alternatively you can enter the EDIT mode directly from the command mode by typing in

EDIT [RETURN]

This command may be abbreviated to ED and if you wish to select the EDIT

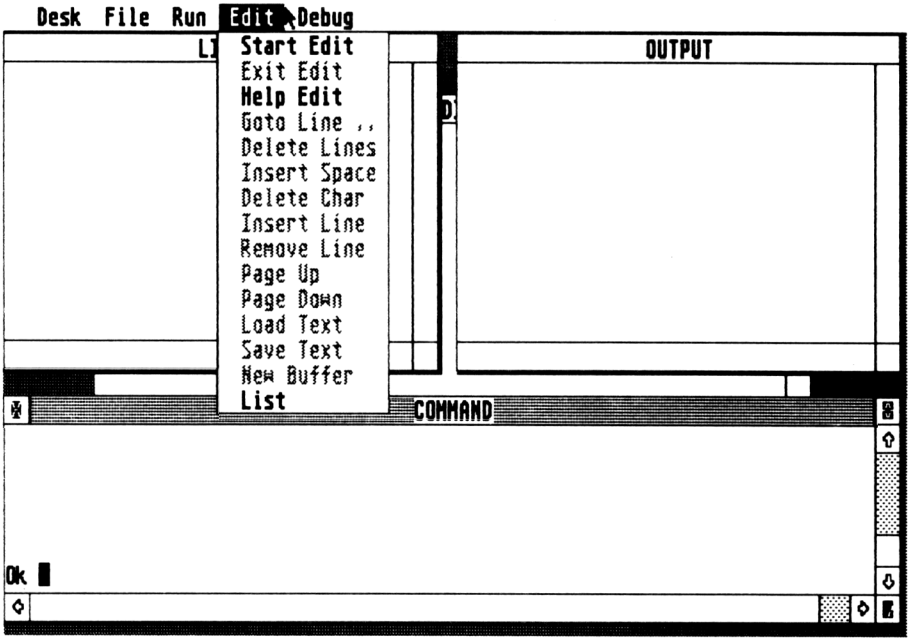


Figure 2.3 *Selecting the Edit mode: screen appearance after clicking on Edit*

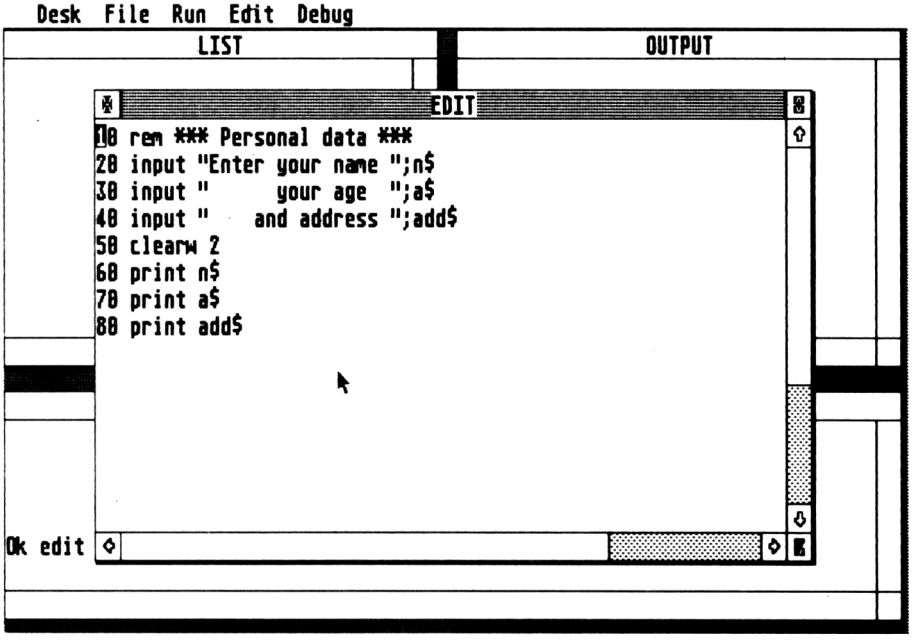


Figure 2.4 *Clicking on the Start Edit option to bring down the Edit window*

mode and start editing at a given line, type in,

ED line number [RETURN]

For example,

ED 300

will bring up the edit window with the cursor at the beginning of line 300, ready for immediate editing.

You use the cursor control keys to position the cursor where you want. Note as you move the cursor to a given line and make changes, the full black type style changes to a "ghost" image. This indicates that the changes have not yet been stored in the computer's memory. As soon as you press RETURN the style changes back to the original, signifying the edited line has been stored, and the cursor moves to the next line.

The F1 to F10 keys on the computer keyboard provide the edit functions. These are summarised under Help Edit; select the Edit menu, click on Help Edit and the list will drop down, as shown in Figure 2.5. Click on the OK box and the Help Edit list will disappear and you will be returned to the Edit window.

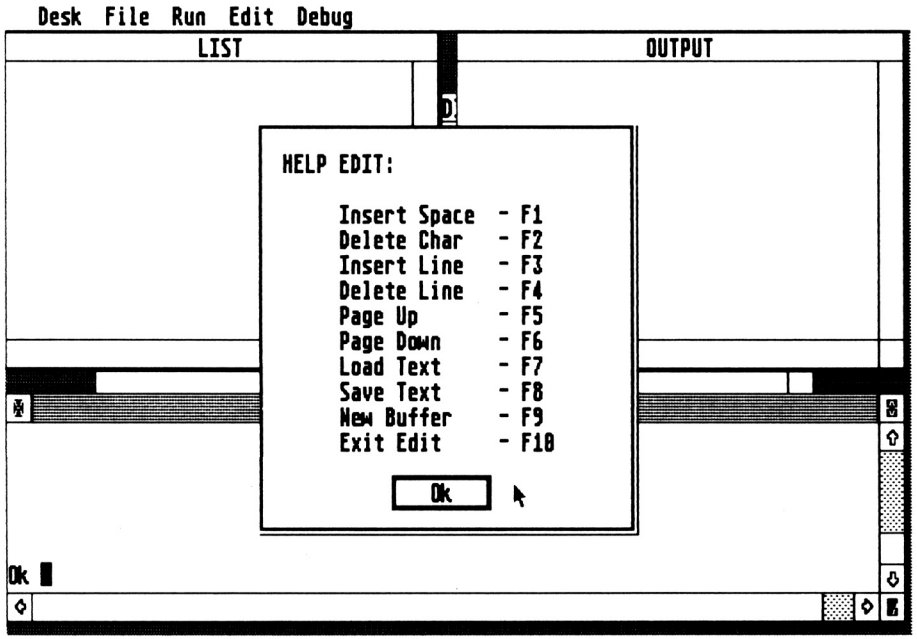


Figure 2.5 *The Help Edit list explaining the action of the F function keys*

Thus, for example, if you wish to delete a line move the cursor down to that line and press F4. The line will change to the ghost style and remain as such even after pressing RETURN. To see what is now actually stored in memory press F9, the New Buffer key. This will display your edited program in the Edit window, in this case with the line removed. To leave the Edit mode and to return to the Command mode, press F10, the Exit Edit key.

Saving programs on disk

Saving the program currently stored in the computer to a disk is very simple. Place a non-write-protected disk, which has already been formatted, in the disk drive unit. Formatting itself is easy. If the disk you are using has not been formatted this must first be accomplished before loading the ST BASIC disk. To format, select the File menu immediately the computer has booted itself up and click on Format and follow the straightforward instructions. Write-protection can be removed by sliding closed the small black plastic switch, situated in the bottom left-hand corner of the disk, i.e. so you cannot see through the small window.

Thus with a formatted, non-write-protected disk in the drive unit and with ST BASIC loaded, select the File menu and clock on the Save As option [see Figure 2.6(a)]. Immediately the ITEM SELECTOR window will be displayed [see Figure 2.6(b)]. Now type in the name under which your program is to be saved. This name must start with an alphabetic letter and contain a maximum of eight characters before the full point and three for the extension to the right of the full point. As you type, your file-name will appear under Selection as shown in Figure 2.6(b). Note that if you do not type in any characters for the extension, but just press RETURN, your program will be saved and given the extension BAS, short for BASIC. Your program can be saved either by pressing RETURN as just described or by clicking with the mouse arrow on the OK box. The ITEM SELECTOR window will disappear and a copy of your program will be saved on disk under your selected name.

You can also save programs using the SAVE command. Simply type in

SAVE file-name [RETURN]

There is, however, an important difference between the SAVE command and the Save As option. The Save As option will write over and therefore replace an existing file saved under the same file-name. When you save using the SAVE command any existing file on disk of the same file-name will not be deleted.

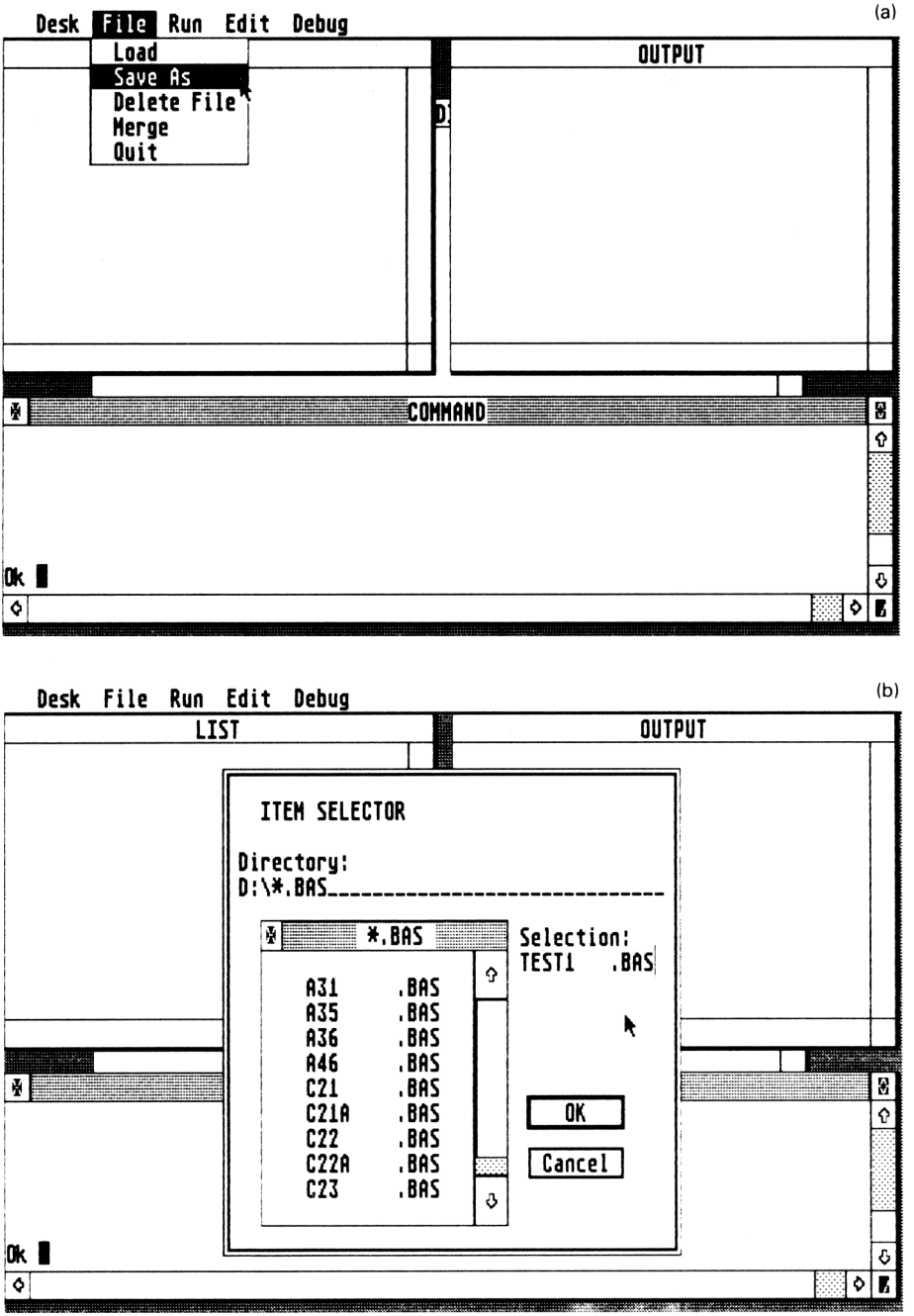


Figure 2.6 *Saving programs on disk: select File on the menu line and then click on the Save As option, see* (a)*; the Item Selector window will appear, type in the file-name and click on the OK box, see* (b)

Loading programs from disk

With ST BASIC loaded and the disk with the required programs on the disk drive, select the File menu and click on the Load option. The ITEM SELECTOR window will immediately drop down showing a list of up to nine programs stored. If the program you wish to load is on the list, simply position the mouse pointer on the file name, click on it (the name will immediately become "highlighted") and move to the OK box and click again. The program selected will then proceed to load. If the required program is not on the current ITEM SELECTOR list click on the upward or downward arrows to move the list contents until you find the file name. Alternatively, and usually quicker, type in the file name followed by either clicking on the OK box or pressing the RETURN key.

Programs can also be loaded directly from the command mode using the command:

LOAD file-name [RETURN]

for example,

LOAD TEST1.BAS [RETURN]

would cause the program stored under the name TEST1.BAS on disk to be loaded from the relevant disk in the drive unit to the computer.

The DIRectory command

To check what files (programs) you have stored, insert your disk in the drive unit and type in

DIR [RETURN]

for a single disk drive unit or

DIR A (or B) [RETURN]

for a dual unit and depending whether your disk is in drive A or B. The DIR command lists all files stored on the disk and displays them in the command window. If you wish to display only a certain group of files you can use the * symbol as a wildcard, e.g.

DIR *.BAS

would list all files stored that have the extension BAS;

DIR C2*.BAS

would list all programs starting with C2 and having the extension BAS.

Merging programs

It is often very convenient especially with longer programming tasks to write and test the program in individual parts and finally merge them into a single complete program. This can be accomplished using the MERGE command. For example, suppose we have written and checked lines 10 to 100 of the first part of a long program and saved it on disk as PART1.BAS. Then after completing the second part, say contained in lines 110 to 250, we can merge this with PART1.BAS as follows. Select the File menu and click on the Merge option. Then select program PART1.BAS on the ITEM SELECTOR list display, or type in PART1.BAS, and click on the OK box. The second part, lines 110 to 250, will then be merged with PART1 and the two parts lines 10 to 250 will now be stored in the computer's memory.

Alternatively you can type in the MERGE command in the command mode, e.g.

MERGE PART1.BAS [RETURN]

will merge program PART1.BAS stored on disk with the current program in the computer's memory. Care must be taken when using MERGE that the line numbers of the parts to be merged do not overlap. If overlap does occur the program saved on disk will erase and replace those lines in the program currently stored when merging takes place.

Finally, the merged program parts can be saved again using the File menu and this time clicking on the Save As option or using the SAVE command direct, e.g. if you wish to save the original PART1.BAS then the merged program would be saved under the name PART12.BAS by,

SAVE PART 12.BAS [RETURN]

Deleting programs stored on disk

To delete a program stored on disk, select the File menu and click on Delete File and then on the file you wish to be deleted on the ITEM SELECTOR list and finally on the OK box. Alternatively you can use the ERA command,

ERA program name [RETURN]

for a single disk drive unit or

ERA A (or B): program name [RETURN}

for a dual unit. For example,

ERA TEST1.BAS [RETURN]

would erase program TEST1.BAS from the disk in a single drive unit;

ERA B:TEST2.BAS [RETURN]

erases program TEST2.BAS in drive B.

Debugging programs

ST BASIC provides an easy means of tracing through a program, either line by line or over specified lines, thus enabling us to obtain an insight as to its working and providing us with a valuable aid to checking for errors. These trace-type operations can be brought into action by either using direct mode commands or the Debug menu.

In the case of the latter, first select Debug on the top line of the screen with the mouse pointer arrow. The Debug menu will immediately drop down [see Figure 2.7(a)]. The Trace and Tron options allow us to trace through a program on a line by line basis; Untrace and Troff are used, respectively, to turn them off.

The Trace option traces through the program currently stored in the computer line by line. Each line is displayed fully in the command window and if any output is produced by the line this will be displayed in the output window after the line has been displayed and executed. Tron functions essentially the same, but displays only the line number (not its content) in the command window, followed by any display of results in the output window.

To run the trace option select the Debug menu, click on the Trace heading in the list; the TRACE window will immediately drop down [see Figure 2.7(b)]. If you wish to trace specified lines type in the line numbers, separating each by a comma or, for a range, the two numbers by a hyphen (minus sign); click on the Lines Entered box and then the OK box. The ALL box is initially high lighted on calling up the Trace option, so if you wish to trace all lines, just click on the OK box. Whether or not you are tracing all or specified lines, you must now either type in RUN [ENTER] or select the Run menu and click on the Run option.

You can, of course, type in the TRACE and TRON commands directly from the keyboard, e.g.

TRACE 30-50 [RETURN]
RUN [RETURN]

will effect a trace of the program line numbers in the range 30 to 50; lines in this range will be displayed in the output window preceded by "t" to denote the trace mode and any output produced by the program will appear in the output window;

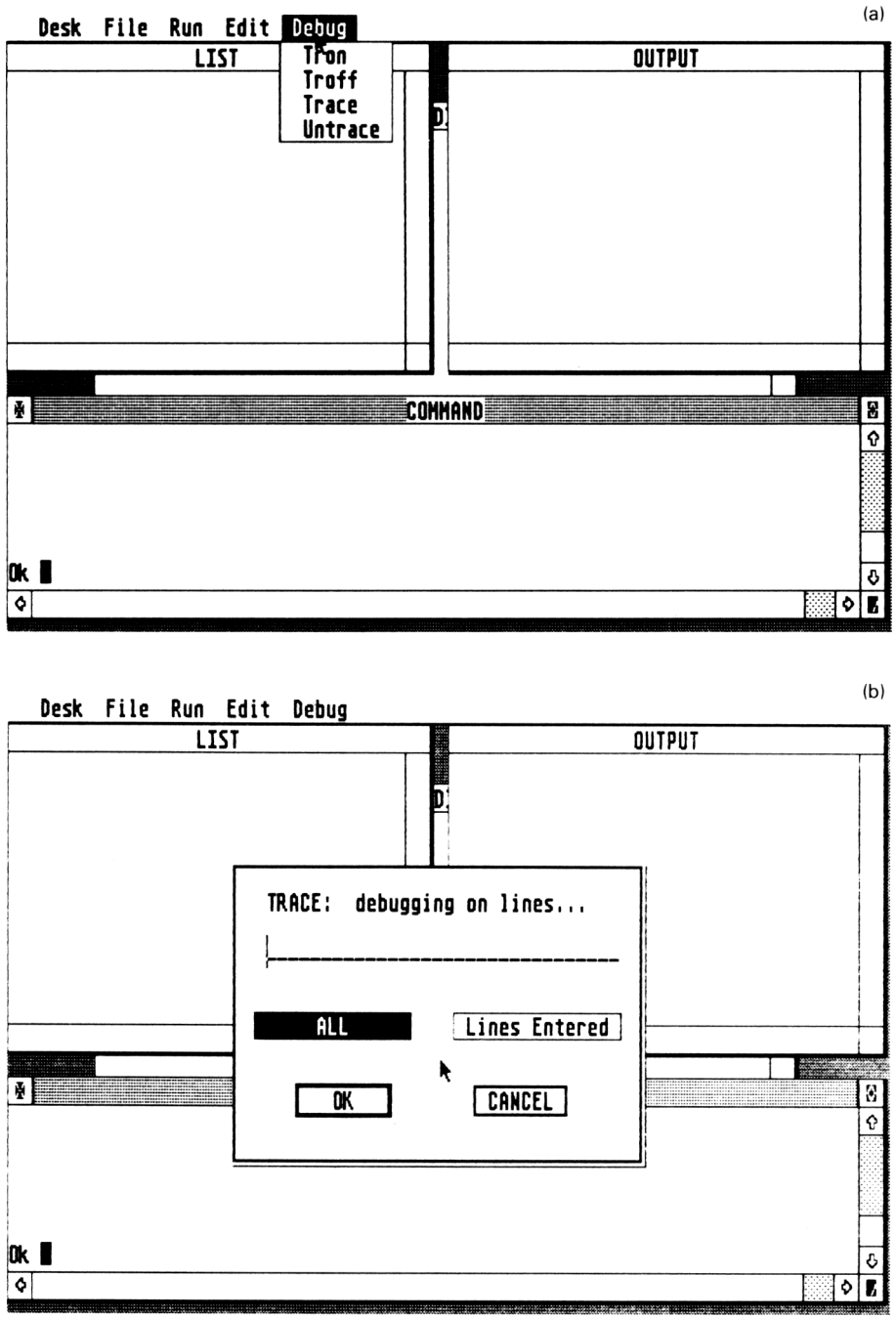


Figure 2.7 *The Debug menu* (a) *and Trace and Tron options* (b) *for checking programs.*

UNTRACE [RETURN]

will switch off the trace function.

TRON [RETURN]
RUN [RETURN]

will effect a trace of the complete program, but only line numbers will be displayed, in square brackets to denote the Tron mode, in the command window; after each line number is displayed any output is also displayed in the output window.

Try running the following program in the trace mode to obtain an idea of what is happening as the program is executed:

```
10      rem *** Test TRACE program ***
15      clearw 2
20      n=n+1:? "Trace of line 20 time ";n
30      print n;" squared = ";n*n
40      if n < 4 then goto 20 else goto 50
50      end
```

TRACE [ENTER]
RUN [ENTER]

The following appears step by step in the command window:

```
t 10    rem *** Test TRACE program ***
t 15    clearw 2
t 20    n=n+1:? "Trace of line 20 time ";n
t 30    print n;" squared = ";n*n
t 40    of n < 4 then goto 20 else goto 50
t 20    n=n+1:? "Trace of line 20 time ";n
t 30    print n;" squared = ";n*n
t 40    if n < 4 then goto 20 else goto 50
t 20    n=n+1:? "Trace of line 20 time ";n
t 30    print n;" squared = ";n*n
t 40    if n < 4 then goto 20 else goto 50
t 20    n=n+1:? "Trace of line 20 time ";n
t 30    print n;" squared = ";n*n
t 40    if n < 4 then goto 20 else goto 50
t 50    end
```

whilst after successive actions of lines 20 and 30, the following is displayed in

the output window:

```
Trace of line 20 time 1
 1 squared = 1
Trace of line 20 time 2
 2 squared = 4
Trace of line 20 time 3
 3 squared = 9
Trace of line 20 time 4
 4 squared = 16
```

EXERCISE PROBLEMS 2.1

1 Write programs to display the following:
(a) names and dates of birth of family and friends
(b) a list of common items and their prices
(c) a football league table

2 Write a program to calculate the volume of
(a) a rectangular block (cuboid) [see Figure 2.8(a)]
$v = b \times l \times d$, for $b = 2.27, l = 5.74, d = 1.86$
(b) a cylinder [see Figure 2.8(b)]
$v = \pi r^2 h$, for $r = 8.63$, $h = 12.4$ and where $\pi = 3.142$.

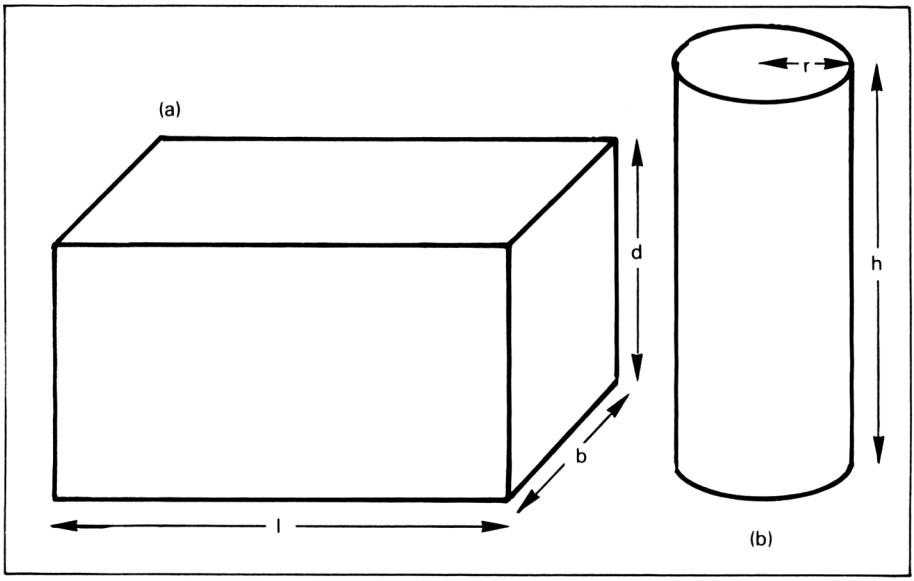


Figure 2.8 *Figure for exercise problem 2:* (a) *cuboid, volume = bld;* (b) *cylinder, volume* $= \pi r^2 h$.

3 Write a program that displays the results of the following calculations:
$3.4 \times 5.67 - 6.21 \times 4.98$,
$5.37^2 + 4.46^2$,
pv^c, when $p = 525$, $v = 72$, $c = 1.4$

4 Write a program to find the average value of the following weights: 20.7 kg, 14.5 kg, 12.9 kg, 13.6 kg.

5 Write a program to calculate the compound interest on
(a) £1000 invested for 5 years at 11%
(b) £25 invested for 32 years at 7.5%.
The compound interest formula is
$I = P(1 + R/100)^N - P$
where I = interest, P = sum invested or principal, R = rate %, N = number of years invested

6 Save the following program as P1.BAS on disk:

```
10      rem *** First part P1 of program ***
20      print "My name is Colin"
30      print "My address is: "
40      print "Deans Hotel, Peshawar"
```

Run the program to check it out before saving it. Now type in NEW [RETURN] and check that the program has been removed from the computer's memory using the LIST command. Type up the second part given below and merge it with P1.BAS using the MERGE command. Save the merged program as P.BAS.

```
10      clearw 2
50      print "The phone number is 79971"
60      print "and I'm only 20 miles from"
70      print "the Khyber Pass and Afghanistan."
80      print "It's hot, over 35 C today"
```

Type in NEW RETURN again and check the program has been cleared using LIST. Load P.BAS and check that both parts have been merged. Note new line 10 replaces the old line 10 in P1.BAS and check that you have done so.

CHAPTER 3

INTERACTING WITH YOUR PROGRAM: DATA INPUT

Introduction and summary

So far the programs we have considered have contained all the necessary information within the actual program with no need to reference any outside source to complete their relatively simple tasks. To enable us to interact with a program during its execution and, for example, be able to enter values from the keyboard, BASIC provides the INPUT statement. We consider the application of INPUT statements in this chapter.

To enable us to store a number of values such as a list of data, BASIC provides the READ-DATA statements. The use of these statements is normally much more convenient than writing individual assignment statements for each variable, especially for a long list of items. READ-DATA statements are also considered.

Frequently we may wish to STOP program execution and make, for example, a check before allowing it to continue. The use of STOP statements and the CONTinue command is considered. In using READ-DATA statements we may also wish to go back to the beginning of the DATA list at some point in the sequence of program execution; to do this BASIC provides the RESTORE statement, the use of which we finally consider.

INPUT statements

INPUT statements are used to input values to the computer directly from the keyboard. They have the basic format:

20 INPUT variable identifier
20 INPUT "...string message..."; variable identifier

The use and action of INPUT statements is illustrated in the following program examples.

Squares and cubes

This program finds the square and cube of any number you input from the keyboard.

```
10      rem *** to find square and cube of ***
20      rem ** any number n INPUT from keyboard **
30      INPUT n
40      print n*n,n^3
```

When we execute the program by typing

RUN [ENTER]

the computer steps over the REMark statements 10 and 20 and executes 30, the input statement. At this point a ? and the cursor appear in the output window indicating that the computer is waiting for us to enter in the value we wish to assign to n. Enter a number, 42 say, and press either the RETURN or ENTER key, i.e. ? 42 [RETURN] or [ENTER].

Program execution continues immediately and line 40 computes and displays the results for 42^2 and 43^3:

1764 74088

The following version makes the program much more informative to the user by including a string message in the input statement:

```
20      clearw 2
30      INPUT "Enter number, n";n
40      print "n squared =";n*n
50      print "n cubed   =";n^3
```

On running the program, line 30 now instructs the computer to display the words within the quotation marks (the string message) followed by ? and the cursor, i.e.

Enter number, n = ?

Now enter in a number followed as always by pressing the [RETURN] key, the results for n^2 and n^3 will be displayed immediately on the next line of the screen.

Conversion calculator
This program acts as a conversion calculator for converting the weight in kilogrammes input from the keyboard to pounds:

```
10      rem *** To convert kilos to lbs ***
20      input "Enter weight in kilos";kg
30      print kg;" kilos = ";kg*2.2046; "lbs"
```

On running the program:

RUN [ENTER]

Enter weight in kilos? 24 [RETURN]. . .note 24 entered by us
24 kilos = 52.9104 lbs . . . result displayed on output screen

Temperature conversion
This short program converts degrees Celcius to degrees Fahrenheit:

```
10      rem ** Degrees C to F **
20      input "Enter temp in deg C";C
30      print C;"deg C = ";C*9/5+32;" deg F"
```

Inputting more than one value
We can input more than one value by either using a series of input statements, e.g.

```
5       clearw 2
10      input "Enter value for A";A
20      input "Enter value for B";B
30      input "Enter value for C";C
40      print "A+B+C =";A+B+C
```

or, more concisely, using a single input line statement:

```
5       clearw 2
10      input "Enter values for A,B,C";A,B,C
20      print "A+B+C = ";A+B+C
```

On running the first version, the program execution pauses at each of the line statements 10, 20 and 30 waiting for us to enter values for A, B, C, respectively, i.e.

RUN [RETURN]
ENTER VALUE FOR A? 47 [RETURN] . . . (47 entered for A)
ENTER VALUE FOR B? 42 [RETURN] . . . (42 entered for B)
ENTER VALUE FOR C? 18 [RETURN] . . . (18 entered for C)
A+B+C = 107 . . . result displayed in output window

On running the single input line version, the computer pauses at line 10 and we enter the values for A, B and C in order and separate each value by a comma, i.e.

RUN [RETURN]
VALUES FOR A, B, C = ? 47, 42, 18 [RETURN]
A+B+C = 107 . . . result displayed

Volume calculations

To find the volume of (for example) a cylinder, we can use INPUT statements to input the dimensions.

```
10      rem *** Volume of cylinder ***
20      clearw 2
30      pi=3.141593
40      input "Enter radius";r
50      input "Enter height";h
60      print "Volume = ";pi*r*r*h
70      print "----------------------"
80      print "for r = ";r,"h = ";h
```

RUN [RETURN]
Enter radius, r = ? 48 [RETURN]
Enter height, h = ? 5.5 [RETURN]
Volume = 36191.15
for r = 48, h = 5.5 . . .display of results obtained

Inputting string variables

So far we have only input numbers. INPUT also works equally well with string variables. For example,

```
10      clearw 2
20      input "Enter your name please";n$
30      input "and your address ";a$
40      input "and your telephone no.";t$
50      print:print
60      print "Information given:"
70      print "name:"tab(15);n$
80      print "address:"tab(15);a$
90      print "telphone:"tab(15);t$
```

Try running this short program. Input your name after the first prompt and address after the second. Immediately you will see them displayed in the output window.

READ-DATA statements

If we wish to use data from within the program rather than input values from the keyboard, which for a long list may tend to be very tedious, we can use READ-DATA statements. For example, suppose we wish to incorporate a list of prices of several items in a program, we can utilise READ-DATA statements as follows:

```
100    READ itemname$,number%,cost,stock.no$
110    DATA desk,132,76.90,D-1886
```

The values of the variables itemname$, number%, cost, stock.no$ in the READ statement 100 are assigned the values in the exact corresponding order as given in the DATA statement of 110. So after execution of line 100, data would have been taken from the DATA statement and assigned as follows:

itemname$ = desk, number% = 132, cost = 76.90
stock.no$ = D-1886

(Note: do not enclose string variable values within quotation marks in DATA statements.)

Each READ statement in a program consists of READ followed by a list of variable identifiers, each identifier being separated from the next by a comma.

Each DATA statement is a list of values and/or expressions, each value separated being separated by a comma. DATA statements may be put anywhere you wish in a program – normally at the beginning or at the end if this is convenient – since the computer ignores any DATA statements until it meets a READ statement.

The first time the computer in executing the program meets a READ statement it takes for the first variable value the first data value from the DATA list, for the second variable the second data value, and so on, working its way through the whole of the DATA statement lists.

If the number of items in the READ and DATA statements do not match, e.g. if there are more variables in the READ statements than values in the DATA statements, then the computer will display an error message immediately it tries to READ a value which is not there.

Program examples for READ-DATA statements

The following program illustrates the basic function of READ-DATA state-

ments:

```
10      clearw 2
20      read A,B,C
30      print "A is asssigned first data item,"
40      print "B the second and C the third"
50      print "i,e, A =";A,"B =";B,"C =";C
60      data 111,222,333
```

RUN [RETURN]
FIRST TERM A = 111
SECOND TERM B = 222
THIRD TERM C = 333

is the display obtained in output screen window showing that A is assigned the first value in the DATA statement list, B the second and C the third.

The following program works out the average value of a number of items listed in the DATA statements:

```
10      rem * To find average of list of values *
20      clearw 2
30      read a,b,c,d,e
40      sum =a+b+c+d+e
50      print "sum of first 5 terms ";sum
60      read f,g,h,i,j
70      sum =sum+f+g+h+i+j
80      print "sum of first 10 terms =";sum
90      read k,l,m,n
100     sum =sum+k+l+m+n
110     print "total sum, all 14 terms =";sum
120     print "Average =";sum/14
130     data 23,87,90,65,43,6,82
140     data 55,87,52,68,8,29,80
```

RUN [RETURN]
sum of first 5 terms 308
sum of first 10 terms 590
total sum, all 14 terms 775
Average = 55.3571

The following program uses read-data statatements to assign data to a list of items. It then works out sub-total and total cost of all items and displays the list and results on the screen.

```
10      rem *** List cost program ***
20      clearw 2
25      fullw 2
30      print "item";tab(12)"number";
40      print tab(20)"unit-cost";tab(33)"sub-total"
50      print"-------------------------------------------"
60      read item1$,n1%,unit.cost1
70      cost1=n1%*unit.cost1
80      print item1$; tab(13)n1%;
90      print tab(20)unit.cost1;tab(33)cost1
100     read item2$,n2%,unit.cost2
110     cost2=n2%*unit.cost2
120     print item2$;tab(13)n2%;
130     print tab(20)unit.cost2;tab(33)cost2
140     read item3$,n3%,unit.cost3
150     cost3=n3%*unit.cost3
160     print item3$;tab(13)n3%;
170     print tab(20)unit.cost3;tab(33)cost3
180     print
190     sum=cost1+cost2+cost3
200     print "Total cost =";sum
202     print"-----------------------"
210     data desks,1264,57.20
220     data chairs,2341,23.76
230     data cabinets,234,99.99
```

On running the program you will obtain the following display in the output window:

```
item          number   unit-cost      sub-total
-------------------------------------------------
desks           1264     57.2           72300.8
chairs          2341     23.76          55622.1
cabinets        234      99.99          23397.6

Total cost = 151320
-----------------------
```

STOP statement and CONtinue command

Frequently we may wish to STOP program execution at a given line and, for example, make a check on the results so far obtained before allowing the execution to continue. This is especially useful in checking intermediate results in a lengthy program. For STOPPing execution of a program at a given line, BASIC provides the STOP statement:

100 STOP

This statement STOPs execution of the program at line 100 and will return the computer to the command mode.

The CONTinue command allows a program that has been interrupted by a STOP statement to be resumed at the next line after the STOP statement. For example, consider the following:

```
10      rem *** STOP and CONTinue demo ***
20      clearw 2
30      read A,B
40      print (A-B)/(A*A-A*B+6*B*B)
50      stop
60      print "Execution continues..."
70      read x,y,z
80      print x^2.3+6.8*y-z*z:stop
90      data 23.8,78.4,21,,45.9,6.7,88.6
```

On running the program execution starts from the beginning, READs in DATA for A,B at line 30, outputs results at line 40 and then is stopped by the STOP statement at line 50. Program execution can be restarted from line 60 by typing in the direct mode command CONT. A further READ is made at line 70, results are output at line 80 and then execution is also stopped at line 80 by a second STOP statement.

RESTORE statement with READ-DATA

In the "normal" use of READ-DATA statements the computer works from the beginning of the first DATA statement (the one with the lowest line number) when it meets the first READ statement and then subsequently works through all the DATA lists until it reaches the end.

This sequence can be "restored" by using RESTORE statements which have the form:

100 RESTORE
100 RESTORE line number

The first RESTORE statement (with no line number after RESTORE) forces any subsequent READ statement to start reading data from the first value of the first DATA statement (lowest number) in the program. When a specified line number is given after RESTORE, then any subsequent READ statement takes its data from the first DATA statement at or immediately following the specified line number in the RESTORE statement.

For example, let us work through the following program:

```
10      rem *** restore demo ***
15      clearw 2
20      read a,b,c
30      print a,b,c
40      restore
45      print "read pointer restored to beginning"
50      read a,b,c,d,e,f
60      print a,b,c,d,e,f
70      data 11,22,33,44,55,66,77,88,99
```

RUN [RETURN]
11 22 33 . . . (display of a,b,c at line 30)
11 22 33 44 55 66 . . . (display of a,b,c,d,e,f at line 60)

At line 20 the READ statement assigns a = 11, b = 22 and c = 33, i.e. the first three values in the DATA list of line 70. Thus at line 30 we obtain the display:

11 22 33

Line 40, the RESTORE statement, returns us to the beginning of the DATA list. Thus at line 50, the READ statement assigns a = 11, b = 22, c = 33, d = 44, e = 55, f = 66 and this is verified by the output at line 60.

Finally, consider a second example:

```
10      rem *** Second restore demo ***
20      clearw 2
30      read a,b,c
40      print a;b;c;(a+b+c)/3
50      restore 2000
60      read a,b,c,d
70      print a;b;c;d;(a+b+c+d)/4
80      restore 1000
90      read a,b,c,d,e,f,g
100     read h,i,j,k,l,m
110     print a;b;c;d;e;f;g
120     print h;i;j;k;l;m
1000    data 1,2,3,4,5,6
2000    data 10,20,30,40
3000    data 100,200,300
```

```
RUN [RETURN]
1 2 3 2
10 20 30 40 25
1 2 3 4 5 6 10
20 30 40 100 200 300
```

is the display obtained.

At line 50 the RESTORE statement "restores" the read pointer to the first value in DATA statement 2000. Thus the READ statement at line 60 assigns A = 10, B = 20, C = 30, D = 40. In the absence of 50 RESTORE 2000, DATA would have been taken from the next value in line 1000, i.e. A = 4, B = 5, C = 6 and then from line 2000, D = 10.

At line 80, the RESTORE 1000 statement returns the read pointer to the beginning of DATA list 1000 so the READ statement 90 starts reading in its data from the first value in 1000 and reads the first seven values. The second READ statement 100 continues to read data working through the DATA lists 2000 and 3000. The data read in is output by the PRINT statements 110 and 120.

EXERCISE PROBLEMS 3.1

1 Write a program that requests you to enter your name, age and address and then displays these on separate lines in the output window.

2 Write a program using READ DATA statements to find the average value of the following 20 numbers:

52 14 37 67 73 11 75 89 19 24
61 49 12 33 47 55 16 71 81 92

3 The following program using the INPUT statement could also be used to find averages:

```
10      rem *** to find average ***
20      rem  ** values input from keyboard **
30      input "Enter value";n
40      count=count+1
50      sum=sum+n
60      print "average so far =";sum/count
70      goto 30
```

Try running the program.

This program has a GOTO statement at line 70 and in fact will never stop! This statement "jumps" execution back to 30 so the whole sequence 30, 40,

50 and 60 is repeated, followed again by the GOTO 30 at line 70 which returns execution to 30, forming a closed loop; bad programming practice! When you have finished entering values you can BREAK the loop and return to the command mode. BREAK can be accomplished by selecting the Run option on the menu bar and checking with the pointer on Break. We can exit the program now by typing in STOP (or END) which transfers control back to the Command window.

4 Using INPUT statements, write programs to convert
(a) feet and inches to metres (1 inch = 2.54 cm)
(b) pounds and ounces to kilogrammes (1 lb = 0.4536 kg).

5 Write a program that allows you to enter the centre position and radius of a circle and then displays the circle in the output window.

6 Write a program that allows you to enter the coordinates (A,B),(C,D) and (E,F) of the corners of a triangle and then draws the triangle in the output window.

CHAPTER 4

DECISION-MAKING, REPETITION, JUMPING AND SUBROUTINES

Introduction and summary

So far in virtually all the programs we have considered the computer executes the individual line statements in exact ascending-order sequence, e.g. line statement 10 before 20, 20 before 30 and so on.

We now consider the BASIC statements available for the important tasks of:

- Making decisions as to alternative courses of action and "jumping" with a program.
- Repeating a section of program statements a controlled number of times.
- Repeating a group of statements while some condition is satisfied.
- Selection of one of several different courses of action.
- Using subroutines.

We commence the chapter by explaining the use of IF statements and test expressions used in making decisions. We then consider the FOR and WHILE constructs used for repetition and the use of Boolean logic expressions for forming more comprehensive test condition expressions. Finally we consider the use of GOTO and GOSUB. In all cases the application of these statements is illustrated in practical program examples.

IF statements

The choice of one of two different courses of action in a program is made using statements of the form:

100 IF test expression THEN statement(s) to be executed
100 IF test expression THEN statement line number
100 IF test expression THEN statement(s) ELSE statement(s)
100 IF test expression THEN line no.N1 ELSE line no. N2

The IF...THEN and IF...THEN...ELSE statements are the fundamental "decision-making" statements in BASIC. Flowcharts illustrating their action are shown in Figure 4.1. IF the test expression is satisfied, i.e. yields a TRUE

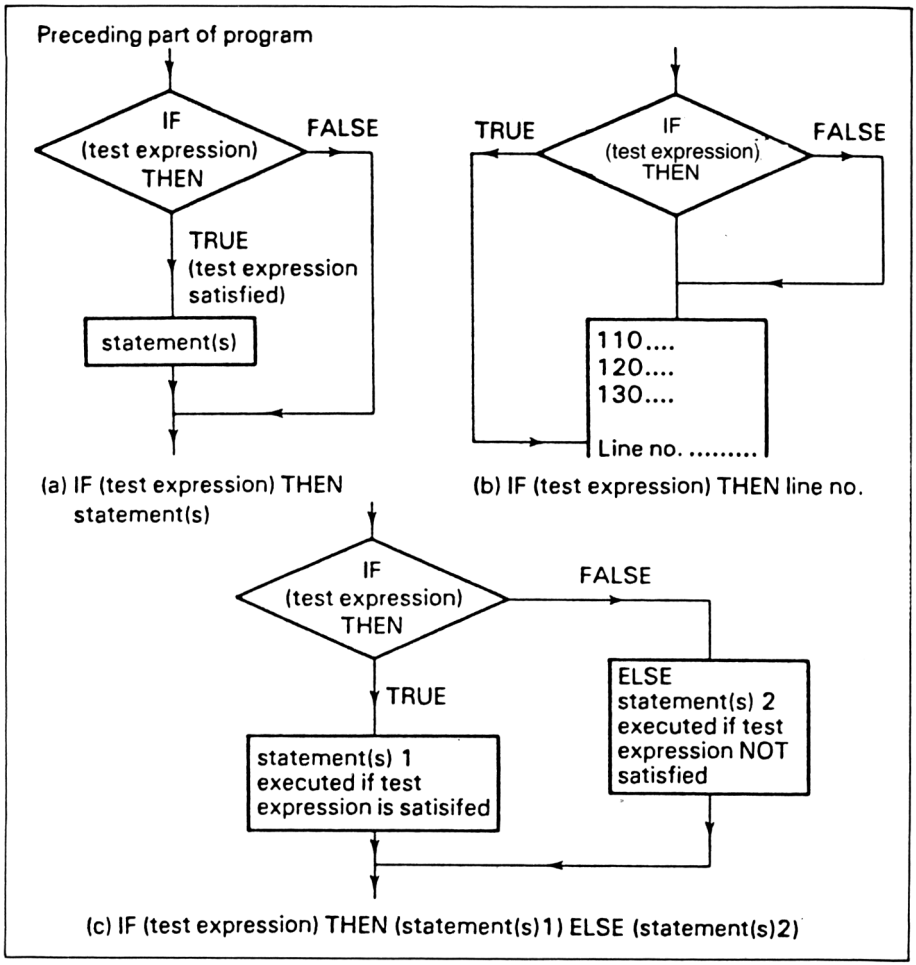


Figure 4.1 *Flowchart representation of IF ...THEN and IF...THEN ...ELSE statements*

value, THEN the statement(s) following THEN are executed; if a line number follows THEN execution is transferred to this line and any interleaving lines are skipped over. The action of the IF...THEN...ELSE statement is very similar. IF the test expression is satisfied THEN the statement(s) following THEN are executed or program execution is transferred to line number "N1" if a line number follows THEN. IF the test expression is not satisfied, i.e. yields a FALSE value, THEN the statement(s) following ELSE are executed or program execution is transferred to line number "N2".

The test expressions used in making the decisions are formed using the comparison operators listed in Table 4.1 below. Comparison expressions can also be used in conjunction with the Boolean operators (AND, OR, NOT – see below) for forming test expressions.

Table 4.1 Table of comparison operators, symbols and meaning

Symbol	*Meaning*
=	equality: e.g. the expression A=B checks whether the value of the left-hand term A is equal to the value of the right-hand term B
<>	inequality: e.g. A<>B checks whether A is unequal to B
<=	less than or equal to: e.g. A<=B checks whether A is less than or equal to B
>=	greater than or equal to: e.g. A=B checks whether A is greater than or equal to B
<	less than: e.g. A<B checks whether A is less than B
>	greater than: e.g. A.B checks whether A is greater than B

Comparison expressions such as A>B, A<B, X<>Y ...etc. always provide one of two results, either TRUE or FALSE.

IF statements: example of use

- This example shows the IF statements being used with a string variable in the test expression. Try running the program. IF you enter red or green or amber the appropriate IF statement will be actioned.

```
10    rem *** if...then demo 1 ***
20    clearw 2
30    input "Enter clour of traffic light";
        light$
40    if light$="red" then print "STOP"
50    if light$="amber" then print "CAREFULL"
60    if light$="green" then print "GO"
```

- A simple example of IF...THEN...ELSE statement:

```
10    rem *** if ... then ... else demo ***
20    clearw 2
30    input "Enter patient's temperature";t
40    if t>102 then 50 else 70
50    print "send for doctor"
60    end
70    print "take 2 aspirin and stay in bed"
```

- A more comprehensive and useful example which you can adapt to write your own program for "finding addresses":

```
5     clearw 2
10    print "Finding friends addresses"
20    print "--------------------------"
30    n1$="Fanny":n2$="Josey
40    n3$="Derry":n4$="Ross"
50    n5$="Helen":n6$="Tom"
60    n7$="Alf":n8$="Beth"
70    print "To find address enter christian
        name"
80    print "Addresses are available for;"
90    print n1$,n2$
100   print n3$,n4$
110   print n5$,n6$
120   print n7$,n8$
130   input "Enter Christian name ";n$
140   if n$="Fanny" then ? "Holly Park N7"
150   if n$="Josey" then ? " 14 Bath Rd,
        Hornford"
160   if n$="Derry" then ? " Rose Cottage,
        Lille"
170   if n$="Ross"  then ? " 22a Block F,
```

```
        Downley"
180   if n$="Helen" then ? " Flat 7, Ave
        de Dusquene"
190   if n$="Tom"   then ? " The George,
        Felpham"
200   if n$="Alf"   then ? " 66 Chanbury
        Close,NW 12"
210   if n$="Beth"  then ? " Deans Hotel,
        Pushto County"
```

Repetition: the FOR loop

FOR type statements, or really FOR loops, provide the means for repeating the actions performed by a group of line statements a controlled number of times. The FOR loop takes the following form:

```
100   FOR variable identifiers = start value T0 end value
110
120   } group of statements to be repeatedly executed
130
170   NEXT variable identifier
```

On entering the FOR loop the variable identifier (control variable for the loop) is set to the start value; the group of statements terminated by NEXT variable identifier are executed with this value and when completed the value is incremented by 1 and the process repeated continuously until the control variable value reaches the end value. By including STEP in the first line, i.e.

100 FOR identifier = start val. TO end val. STEP increment

the control value may be incremented by any value – whole, decimal, positive or negative rather than by just 1.

A flowchart illustrating the action of the FOR loop is shown in Figure 4.2.

FOR loops: examples of their use

- On running the following simple FOR loop program:

```
10    rem *** FOR loop demo ***
20    clearw 2
30    for n=1 to 5
40    print "Tea please !"
50    print "************"
60    next n
```

i.e. RUN [RETURN]

you will obtain the display

```
Tea please !
**********
Tea please !
**********
Tea please !
**********
Tea please !
**********
Tea please !
**********
```

Statement 30 instructs that n should be changed consecutively from 1 to 5 in steps of 1. Thus statements 40 and 50 are each executed five times with n incremented by 1 at the end of each cycle.

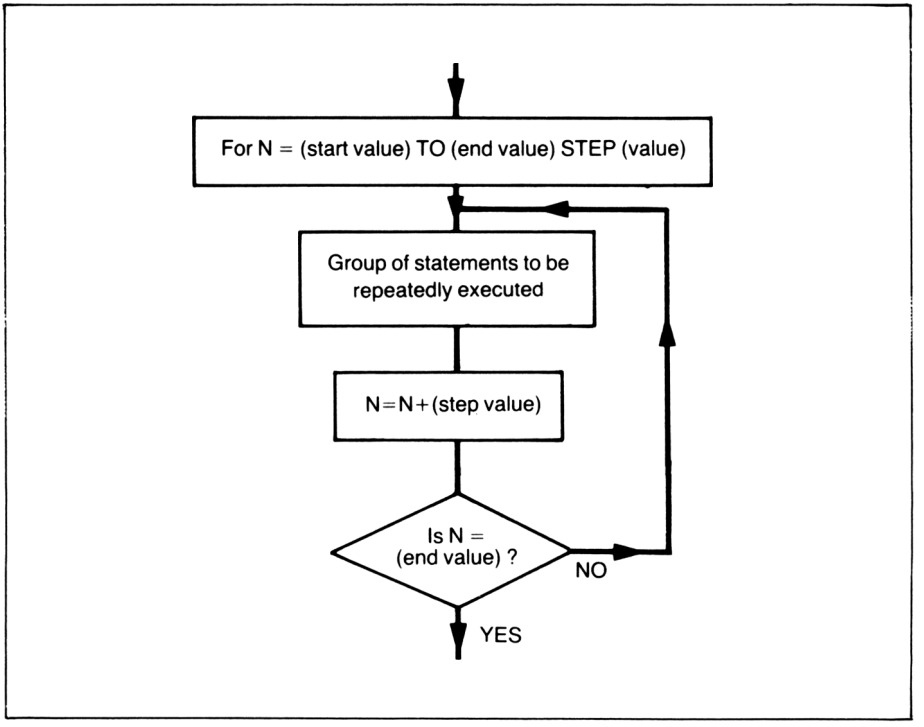


Figure 4.2 *Flowchart representation of FOR loop*

- This program utilises the FOR loop to display a table of squares, cubes and fourth powers of the number from n = 1 to 10.

```
10    rem ** For loop for tabular display **
15    clearw 2:fullw 2
20    print "Squares, cubes and 4th powers"
30    print "-----------------------------"
40    print " n","n^2","n^3","n^4"
45    print
50    for n=1 to 10
60    print n,n^2,n^3,n^4
70    next n
```

On typing, RUN [RETURN] we obtain the display of the table in the output window.

n	n^2	n^3	n^4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561
10	100	1000	10000

- This program illustrates the use of STEP. The FOR loop statement at line 50 uses a 0.1 step value; the second FOR loop at line 120 uses a negative step value of −0.25:

```
10    rem *** For loop with STEP ***
20    clearw 2:fullw 2
30    print "x","x^2","x^3"
40    print "-------------------------------"
50    for x=0.1 to 0.9 step 0.1
60    print x,x^2,x^3
70    next x
80    print
90    print "With negative step value"
100   print "n","1/n"
110   print "-----------------------"
120   for n=2 to 1 step -0.25
```

```
130    print n,1/n
140    next n
```

On running the program, you will obtain the following:

```
x                x^2              x^3
-----------------------------------
 .1               .01              9.99999E-04
 .2               .04              7.99999E-03
 .3               8.99999E-02      2.69999E-02
 .4               .16              6.39999E-02
 .5               .25              .125
 .6               .359999          .215999
 .7               .489999          .343
 .8               .64              .512

With negative step value
n                1/n
------------------------
 2               .5
 1.75            .571428
 1.5             .666666
 1.25            .8
 1               1
```

In the above, note that the computer does not give figures with absolute accuracy!

- This simple program uses the FOR loop to draw 10 concentric circles in the output window

```
10     rem *** Concentric circles ***
20     clearw 2:fullw 2
30     for n=1 to 10
40     circle 300,150,12*n
50     next n
```

A similar display produced with a concentric ellipses program is shown overleaf in Figure 4.3.

- This program utilises the FOR loop to draw up a table showing how a given sum invested at a given rate grows annually over a period of 10 years. The program is set up with:

p = sum invested = £100, r = interest rate, % = 8.25

and uses the compound interest formula:

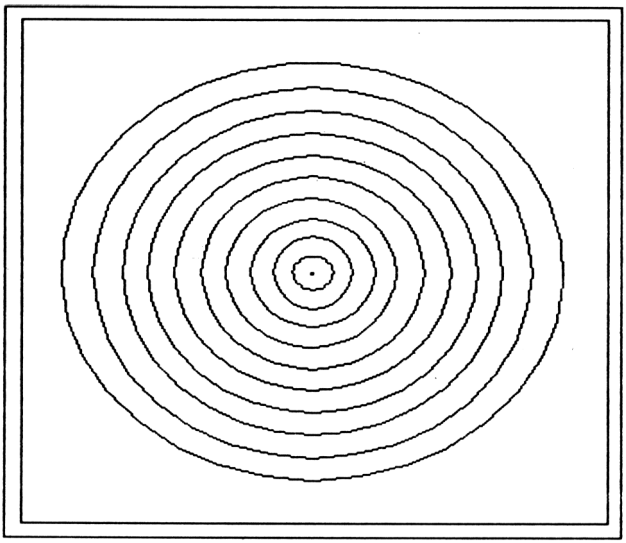


Figure 4.3 *Display produced by concentric ellipses program*

$a = p\,(1 + r/100)^n$

where a = amount = sum invested + interest and n = number of years.

```
10      REM * Compound Interest Calculation *
20      clearw 2:fullw 2
30      print "year","interest","total"
40      print "-------------------------------------"
45      p=100:r=8.25
50      for n=1 to 15
60      a=p*(1+r/100)^n
70      print n,
80      print using "#####.##";a-p;
90      print using "#########.##";a
100     next n
```

On running the program with the above values you will obtain the following display:

```
year            interest        total
-------------------------------------
 1                 8.25          108.25
 2                17.18          117.18
 3                26.85          126.85
 4                37.31          137.31
 5                48.64          148.64
 6                60.90          160.90
 7                74.18          174.18
 8                88.55          188.55
 9               104.10          204.10
10               120.94          220.94
11               139.17          239.17
12               158.90          258.90
13               180.26          280.26
14               203.38          303.38
15               228.41          328.41
```

Try modifying the program so you can input your own values for p and r. For example, include the lines

24 INPUT "Enter sum invested"; p
28 INPUT "and rate of interest"; r

and delete line 45.

Repetition: the WHILE loop

The WHILE...WEND loop provides a very useful form of repetitive control which causes a group of statements to be repeated WHILE a test condition is satisfied.

The WHILE (test condition expression)
The WHILE loop takes the form:

```
110   WHILE (test expression expression)
120  \
130   \
  .    } group of statement(s) to be executed WHILE test expression is satisfied
  .   /
  .  /
170 WEND
180
```

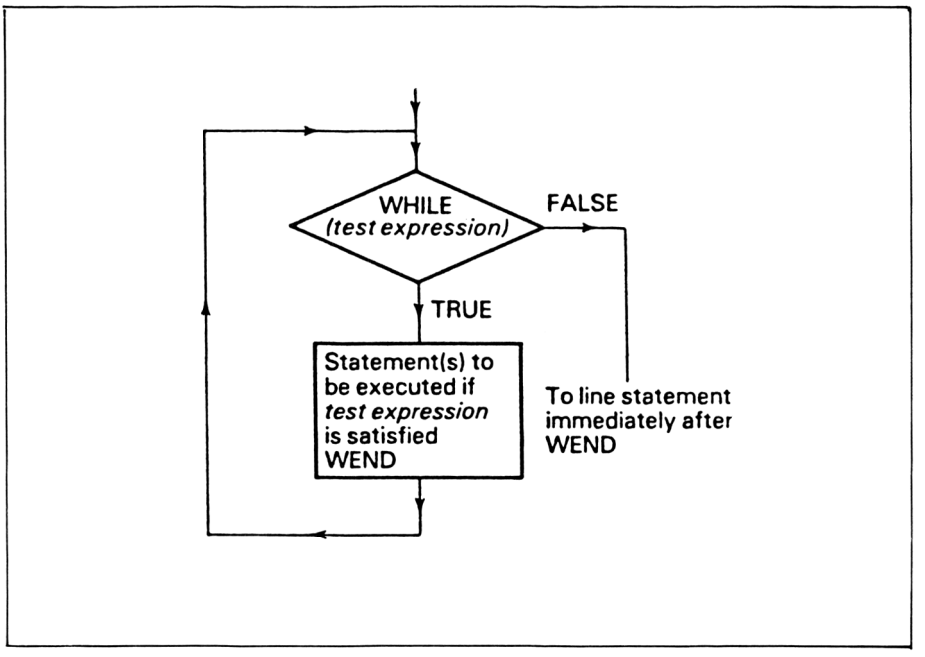


Figure 4.4 *Flowchart representation of WHILE...WEND loop*

The test expression following WHILE determines whether or not the statement(s)grouped between WHILE and WEND are executed. This expression is evaluated at the beginning of each cycle. If it yields a TRUE value the following statements are executed, if not they are skipped and execution transfers to the statement immediately following WEND. Remember the statements between WHILE and WEND should eventually produce a result to alter the test condition expression to FALSE, otherwise your program will be trapped in the loop for "ever"!

Figure 4.4 shows a flowchart representation of the WHILE loop.

WHILE loops: examples of their use

- The following program uses the WHILE...WEND loop to READ fifty marks from the DATA statements. WHILE n<50 it READs each mark and counts those equal to and above 50 and those below 50. After reading the fiftieth mark the WHILE loop is exited and the number of "passes" and "failures" is output.

```
10      rem ***while-wend demo ***
20      clear w 2
30      n=0
```

```
40      while n<50
50      read m
60      if m>=50 then p=p+1
70      if m<50 then f=f+1
80      n=n+1
90      wend
100     print "no. of passes p= ";p
110     print "no. of failures f= ";f
120     data 23,67,54,31,9,67,39,66,83,17
130     data 19,93,12,34,76,56,9,31,45,12
140     data 67,98,34,65,31,9,56,32,45,10
150     data 45,76,43,98,76,3,23,12,72,23
160     data 51,76,56,94,80,6,34,83,44,81
```

- This program uses the WHILE loop to continually sum values entered from the keyboard and display the current sum and average. Exit from the WHILE loop occurs when a zero is entered, see line 60.

```
10      REM *** Running total and average ***
20      clearw 2
30      sum=0:n=0
40      print tab(30)"sum so far","average"
50      input "Enter first value";x
60      while x<>0
70      sum=sum+x
80      n=n+1
90      print tab(30) sum;tab(48)sum/n
100     input "next value";x
110     wend
```

Boolean operators and logic expressions

Boolean-logic expressions are used essentially to form the test expressions for decision-making in IF. . .type statements. A Boolean expression, just like the comparison expressions considered above, can take only one of two values: TRUE or FALSE.

In addition to the comparison operators, four Boolean or logic operators:

NOT, AND, OR, XOR

are used to create Boolean expressions and are very invaluable in forming test condition expressions. Their meaning is explained below.

NOT the logical NOT or logical negation
e.g. NOT A is TRUE if A is FALSE, or FALSE if A is TRUE

AND the logical AND
e.g. A AND B is TRUE if and only if A and B are both TRUE; if A and/or B are FALSE, A AND B is assigned a FALSE value.

OR the logical OR
e.g. A OR B is TRUE if either or both A and B are TRUE; if A and B are both false A OR B is assigned a FALSE value.

XOR the logical EXCLUSIVE OR
e.g. A XOR B is TRUE if either A or B is TRUE; if A and B are both TRUE or both FALSE A XOR B is FALSE.

Precedence

The order of precedence of these operators in evaluating Boolean expressions is as follows:

highest	NOT
	AND
	OR, XOR
lowest	=,<>,<=,>=,<,>

A simple Boolean expression consists of a series of Boolean values separated by AND, OR, XOR or preceded by NOT; for example, suppose A,B,C,D are test expressions or variables which are assigned either TRUE or FALSE values after evaluating comparison type expressions, then:

A AND B AND C AND D is TRUE if and only if A,B,C,D are all assigned TRUE values

A AND NOT D is TRUE if A is TRUE and D is FALSE

A OR C OR D is TRUE if one or more of the variables is assigned TRUE

C XOR D is TRUE if either C or D is TRUE, otherwise it is FALSE.

Examples

- The following program illustrates the meaning of the fundamental AND, OR and XOR operators. Enter various combinations of 0 and 1s for the variables A and B, the results of

  F1 = A AND B
  F2 = A OR B
  F3 = A XOR B are displayed

```
10    rem *** boolean demo ***
20    clearw 2
30    print " Input states A and B"
40    print "are either 0 or 1"
50    print "           -------"
60    input "Enter states for A and B";A,B
70    print
80    print "AND example:"
85    print "------------"
90    F1 = A AND B
100   print "for A =";A;" and B =";B
110   print "A AND B =";F1
120   print "OR example:"
130   print "-----------"
140   F2 = A OR B
150   print "for A =";A;" and B =";B
160   print "A OR B =";F2
170   print "Exclusive OR example"
180   print "--------------------"
190   F3= A XOR B
200   print "When A= ";a;" and B = ";B
210   print "A XOR B =";F3
```

- The example below illustrates the formation of test conditions using a combination of Boolean and comparison operators, see lines 40 and 70. These expressions are used, respectively, in lines 50 and 70 as the test condition expression in IF statements. Try running the program to see how it works.

```
10    rem *** Boolean expression demo ***
20    clearw 2
30    input " Numbers a and b";a,b
40    state.OK = a>10 and b<20
50    if state.OK then ? "O K" else goto 90
60    input "Numbers x and y";x,y
70    if x>0 OR y>=50 then ? "O K" else 90
80    print "All conditions O K ":end
90    print "Conditions not satisfied":end
```

- The following program illustrates the use of a logic expression in the control of a WHILE loop. The program could be regarded as a control simulation exercise for a machine or process, i.e. WHILE A = 1 AND B = 1 AND C = 1 keep the process running but make checks on the A,B,C

control values. If a fault occurs exit from the WHILE loop and display which fault(s) have occurred.

Once again try running the program.

```
10    rem*** Simulation example ***
20    clearw 2
30    print "If A,B,C tests OK enter 1"
40    print "If not, enter 0"
50    print
60    A=1:B=1:C=1:rem initial conditions
        set as OK
70    while A=1 AND B=1 AND C=1
80    print "All conditions OK"
90    print "Continue running"
100   print:print "Test conditions"
110   input "ENTER A,B,C values 1 or 0";A,B,C
120   CLEARW 2
130   WEND
140   IF A=0 THEN ? "FAULT A; CHECK OIL"
150   IF B=0 THEN ? "FAULT B;OVER-HEATING"
160   IF C=0 THEN ? "FAULT C;SAFTEY HAZARD"
```

GOTO commands, statements and use of labels

In many program solutions we frequently wish to "jump" the normal ascending order of line statement execution and transfer to a higher or lower line rather than proceed to the statement immediately following. We have already seen how this is done in IF and WHILE type statements. For example,

100 IF x > 0 THEN 200

would transfer execution to line 200 if x is greater than 0. This statement can also be written including GOTO, i.e.

100 IF x > 0 THEN GOTO 200

which has the identical effect. The inclusion of GOTO, however, is not necessary in IF statements.

GOTO is one of the two BASIC "jump" instructions. It can be used as a direct mode command in a similar manner as RUN line number to start the execution of a program at a given line. For example,

GOTO 200 [RETURN]

would start the execution of a program at line 200.

GOTO can also be used to transfer control within a program. The GOTO statement has the general form

100 GOTO line number or label

On reading this statement program execution would jump to the specified line number or, if a label is given, to the line containing the label. For example,

100 GOTO 500

would cause program execution to jump and proceed at line 500;

300 GOTO calculation
.
.
.
750 calculation: PRINT SQR (b*b-4*a*c)

would cause execution to jump to the line with the label "calculation", i.e. in the above example line 750. Note that the label must always be followed by : (a colon). The use of labels, rather than line numbers, helps in understanding the structure of a program and how it operates.

The GOTO x,y statement

The GOTOXY statement shifts the cursor to a given column,row position on the output window display. For example,

10 GOTOXY 10, 15
20 PRINT "*"

will cause the cursor to be shifted 10 columns across the output window and 15 rows downwards. Thus the "*" will be displayed in the 10th column and 15th row.

Examples of use of GOTO and GOTOXY

- This program utilises both the GOTO and GOTOXY statements to allow you to move a display caption around the output display window. Try running it. Remember your can exit the program using Break, i.e. select Run on the menu bar and click on the Break option. Then type in STOP or END.

```
10      rem *** goto and gotoxy demo ***
20      clearw 2:fullw 2
```

```
30    input "Enter required position";x,y
40    gotoxy x,y
50    print "************************"
60    gotoxy x,y+1
70    print "*   ATARI ST-BASIC     *"
80    gotoxy x,y+2
90    print "*     BY               *"
100   gotoxy x,y+3
110   print "*  R. G. M.            *"
120   gotoxy x,y+4
130   print "************************"
140   rem for loop for short delay
150   for n=1 to 2000 :next n
160   clearw 2
170   goto 30
```

- This program allows you the facility for repeated conversions to convert feet and inches to metres. At line 60 you are asked to input the number of feet and inches from the keyboard. Remember to separate the number of "ft" and "in" values by a comma and as always press either RETURN or ENTER keys to complete your entry. The computer will then effect the calculation and display the result, whilst at line 100:

  100 goto 60

  transfers program execution back to line 60 for another calculation. To exit from the program enter −999 for "ft" and any value for "in". The IF statement at line 70 will then END the program.

```
10    rem *** GOTO example ***
20    rem * for repeated conversions *
30    clearw 2
40    print "Feet and inches to metres conversion"
50    print "-------------------------------------"
60    input "Enter no. of feet,inches";ft,in
65    if ft=-999 then end
70    m=(12*ft+in)*0.0254
80    print ft;" feet ";in;" in. = ";m;" metres"
90    print
100   goto 60
```

A slightly modified version of the above program avoiding the direct use of goto and a −999 type entry to end is given below. Line 100 asks the question "another go, enter y or n"; line 110 acts on the answer.

```
10      rem *** example ***
20      rem * for repeated conversions *
30      clearw 2
40      print "Feet and inches to metres conversion"
50      print "-----------------------------------"
60      input "Enter no. of feet,inches";ft,in
70      m=(12*ft+in)*0.0254
80      print ft;" feet ";in;" in. = ";m;" metres"
90      print
100     input "another go, enter y or n ";a$
110     if a$="y" then 50 else end
```

The ON...GOTO statement

We can also use the ON...GOTO statement:

ON select variable GOTO line no. 1, line no.2, line no.3

ON select variable GOTO label 1, label 2, label 3

which will cause program execution to jump to one of the specified line numbers or line numbers specifed by the respective label names according to the value of the select variable.

Transfer to the first line number specified after GOTO will be made if the value of the select variable value is 1; transfer to the second specified line number if the value is 2..., and so on. Any number of line numbers may be specified provided they can be placed on the same logical line. For example,

100 ON n GOTO 200, 300, 400

If the value of n is 1 then program execution will jump to line 200; if n = 2 execution jumps to 300; if n = 3 then execution jumps to 400; if we input a higher value for n, e.g. 4, program execution would proceed to the line immediately following the ON...GOTO statement.

The following program illustrates a practical example of the use of the ON...GOTO statement:

```
10      rem *** on...goto example ***
20      clearw 2:fullw 2
30      print "Ski holiday tariffs"
40      print "Resorts on file:"
50      print "----------------"
60      print "La Plagne...enter 1 "
70      print "Tigne .....enter 2 "
80      print "Adelboden...enter 3 "
```

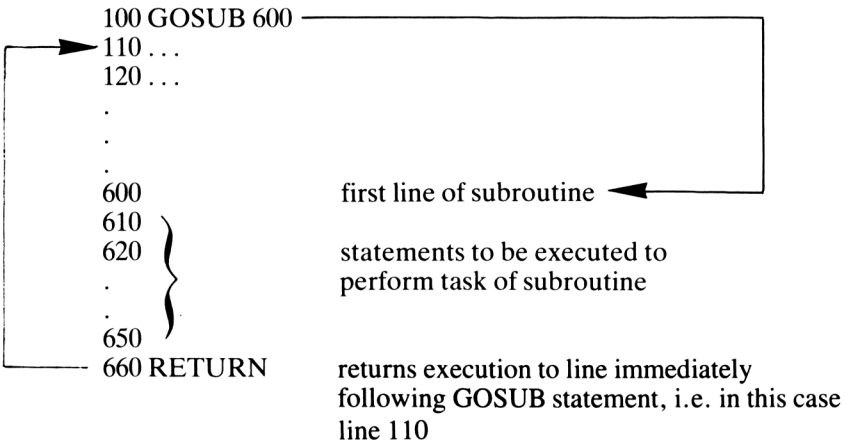
```
90    print "Verbier.....enter 4 "
100   print "Wengen .....enter 5 "
110   print
120   input "Enter selection no. ";n
130   if n>5 then ?"Incorrect entry":goto 40
140   on n goto 150,210,260,280,300
150   LaPlagne:print "La plagne:"
160   print "---------"
170   print "High seasion: FF 3000 per week"
180   print "Low season  : FF 2200 per week"
190   print "Special offers: 8/1 to 29/1 FF1800"
200   end
210   Tigne:print "Tigne:"
220   print "-------"
230   print "HoteL + full board FF 3400 "
240   print "Appartments FF 1500 to 2500"
250   end
260   Adelboden:Print "Adelboden... no vacancies"
270   end
280   Verbier: print "Verbier:"
290   print "High SF 1200, low SF 1050":end
300   Wengen:print "Wengen:"
310   print "Villa Rimalay + half-bd. SF 1100"
320   end
```

The on...goto statement at line 140 causes execution to jump to display the required information; n is used as the select variable, see line 120, and depending whether n = 1,2,3,4, or 5, line 140 directs execution to jump to the appropriate place.

GOSUB statements and subroutines

The action of GOSUB statements are similar to GOTO statements but contain in addition a RETURN. They are used to transfer program execution to a given section of the program which contains a subroutine designed to execute a certain task. After executing the subroutine execution is returned, by a RETURN statement which must form the last line of the subroutine, to the line statement immediately following the GOSUB statement. The use of subroutines have important advantages in programming. The subroutine need only be written once and can be "called" into action as many times as is required in the main program.

The general form of a GOSUB...RETURN construct is given below:


100 GOSUB 600
110 . . .
120 . . .
600
610
620
650
660 RETURN
first line of subroutine
statements to be executed to perform task of subroutine
returns execution to line immediately following GOSUB statement, i.e. in this case line 110

The GOSUB line-number statement transfers processing of a program to the line number specified and is used, for example, to go to a subroutine which follows from that line number. A RETURN statement acts to terminate the subroutine statements and return processing back to the line statement immediately following the GOSUB statement.

In addition to line numbers, label names may also be used.

We can also use the ON. . .GOSUB statement:

ON select variable GOSUB line number-1, line number-2, . . . and

ON select variable GOSUB label name1, label name2,. . .

which is fundamentally the same as the ON. . .GOTO statements considered in the last section. They differ in that program execution always returns to the first statement immediately following the ON. . .GOSUB statement after execution of the subroutine. For example,

```
100 ON X GOSUB 500, 600
110. . . .
.
.
.
500 REM * subroutine 1 *
.
.
.
590 RETURN : REM * End of subroutine 1*
600 REM * subroutine 2*
.
```

.
.
.
750 RETURN : REM* End of subroutine 2*

When X = 1 at line 100, program execution jumps to the subroutine at line 500 and after execution of the subroutine returns to line 110, the line immediately following the ON...GOSUB statement.

If X = 2 at line 100, execution would then jump to execute the subroutine commencing at line 600 and then also return at the end of the subroutine (line 750) to line 110.

The following example illustrates a practical example of using GOSUB. The program plots a horizontal bar chart for the sales figures contained in DATA statement, line 220. The subroutine used to plot bars is contained in the section lines 150 to 210:

```
10    rem * Use of gosub for drawing bar charts *
20    clearw 2:fullw 2
30    print "    5    10    15    20    25 ";
40    print "   30    35    40    45    50 "
50    print "-----------------------------------";
60    print "-----------------------------------"
70    print "U K sales ":gosub barchart
80    print "West Germany":gosub barchart
90    print "South America":gosub barchart
100   print "Phillipines":gosub barchart
110   print "Austria":gosub barchart
120   print "France ":gosub barchart
130   print "Sweden":gosub barchart
140   end
150   barchart:
160   read sales
170   for n=1 to sales
180   print "*";
190   next n
200   print
210   return
220   data 15,35,42,38,25,16,0,7
```

Here is the bar chart display obtained on running the program:

```
    5     10    15     20     25     30     35    40
-----------------------------------------------------
U K sales
***************
West Germany
************************************
South America
********************************************
Phillipines
****************************************
Austria
*************************
France
****************
Sweden
```

EXERCISE PROBLEMS 4.1

1 Write a program that will display a message only if the correct password APK 3287103 is keyed in.

2 Write a program that will print out a standard memo a given number of times. This number is to be input from the keyboard.

3 Write a program that READs up to 50 values contained in data statements and determines their average.

4 Write a program that will select five different courses of action depending on whether 1,2,3,4, or 5 is keyed in.

5 Write a subroutine that will determine the maximum and minimum values in a list of data contained in DATA statements. Incorporate this subroutine in a program which displays these values for the first 10, the first 20 and the total list.

6 Write a program that contains a subroutine which allows entry a single name and address from the keyboard, displays it on the screen and allows any corrections to be made and finally outputs the corrected version to a printer. Incorporate this subroutine in a program that will permit entry of up to 50 names and addresses.

CHAPTER 5

STANDARD FUNCTIONS AND APPLICATIONS

Introduction and summary

A wide variety of standard functions are provided in ST BASIC. In this chapter we describe the form and explain the meaning and use of those functions used in handling numbers, for providing values for mathematical functions and also those used to process characters and strings. We also consider how we can define and use our own "user-defined" functions.

Square roots: SQR(X)

SQR (X), where X is any positive number of expression will compute the square root of the value X within the brackets. For example,

- PRINT SQR (31.36) [ENTER] or [RETURN]
  5.6 square root of 31.36 is displayed on screen

- ```
  10      input "Value of x";x
  20      print "Square root of x =";sqr(x)
  ```

  RUN [ENTER] or [RETURN]

  Value of X = 193.21 [ENTER] . . . note we enter 193.21
  13.9 square root of 193.21 displayed
  ```

```
10      input "Values of a,b,c";a,b,c
20      x=a+b+c
30      y=a*b*c
40      print sqr(x),sqr(y),sqr(x/y)
```

RUN [RETURN]
Values of a,b,c? 2.6, 8.9, 4.7 [ENTER]
4.02492 10.4287 0.385946 results displayed by line 40

We can, of course, use the ^ key to find any power or root (see also Chapter 1:

PRINT 2.25^0.5 gives 1.5, the square root of 2.25
PRINT 46^2 gives 2116 the square of 46
PRINT 112.6^4 gives 1.6075E+08, 112.6 raised to the power of 4
PRINT 59.3^0.68 gives 16.0574, 59.3 raised to power 0.68
PRINT 48^−1.8 gives $48^{-1.8} = 1/48^{1.8} = 9.41382E-04$

ABSolute, INTeger and SiGN of numbers

ABS(X) provides the absolute value, i.e. magnitude, of the value of the number or arithmetic expression X specified within the brackets. For example,

PRINT ABS (44.67) displays 44.67
PRINT ABS (-44.67) displays 44.67, i.e. the magnitude of −44.67
PRINT ABS (32+49) displays 81
PRINT ABS (32−49) displays 17, i.e. $|32 - 49| = |-17| = 17$

INT(X), the INTeger function, converts the real number or expression into an integer by removing any decimal part. For example,

PRINT INT (11.8) displays 11
PRINT INT (-11.4) displays −11
PRINT INT (8.4*6.3) displays 52, as $8.4 \times 6.3 = 52.92$
PRINT INT (−8.4*6.3) displays −52

FIX(X) is similar to INT and truncates a real number or expression to an integer; the application of FIX is limited to the range 32768 to 32767 for the Atari ST series. For example,

PRINT FIX (11.9) returns 11
PRINT FIX (−11.9) return −11

CINT(X) converts the value of X to the nearest rounded whole number in the INTeger range −32768 to 32768. For example,

PRINT CINT (119.72) returns 120
PRINT CINT (−119.72) returns −120
PRINT CINT (98890.87) returns "Overflow" message, "Number too large for an integer".

FLOAT (N) converts an integer into a real number; the number is of the same value but in floating-point form and has more room in memory. N must lie within the range −32768 and 32767.

SGN(X), the SiGN function, is used to determine the sign of the value of the variable or numeric expression X. SGN(X) returns −1 if X is less than 0, 0 if X = 0 and +1 if X is greater than 0, e.g.

SGN (52.6) returns +1
SGN (0) returns 0
SGN (−10.7) returns −1

Random-number generation

RANDOMIZE together with the RND function are used to generate random numbers.

RND and RND(N), where N is an integer, generates a random number in the range 0 to 0.999999. When N is a positive integer or omitted RND returns the next number in the current random generator sequence; when N = 0, i.e. RND(0) returns the last number in the sequence; when N is negative, RND(N) reseeds the random generator and returns the first number in the new sequence.

Using the RND function alone without a RANDOMIZE statement preceding the RND call statement will result in the generation of the same random number list. For example,

```
20      FOR N=1 TO 6
30      PRINT RND
40      NEXT N
```

will produce a list of six "random" numbers which will be the same each time you run the program.

To produce a different list each time, include the RANDOMIZE 0 statement before the RND statement, e.g. the program

```
10      RANDOMIZE 0
20      FOR N=1 TO 6
30      PRINT RND
40      NEXT N
```

will produce a different set of random numbers each time the program is run. The 0 (zero) after RANDOMIZE acts as a "seed" for the random number generator. Try running both programs to see the difference. A typical output

is:

```
.77578
.127475
.93057
.415326
.648335
.207068
```

Using the INT or CINT standard functions you can generate random "whole" numbers in any range (within the integer range of the Atari) as illustrated by the following program:

```
10      REM *** GENERATION OF RANDOM NOS. ***
20      RANDOMIZE 0
30      PRINT"RANDOM NUMBERS: RANGES"
40      PRINT "-------------------------------------------------"
50      PRINT "0 TO 9","0 TO 99","0 TO 999"
60      PRINT "-------------------------------------------------"
70      FOR N=1 TO 10
80      PRINT INT(RND*10),INT(RND*100),
           INT(RND*1000)
90      NEXT N
100     PRINT "-------------------------------------------------"
```

This program generates 10 random numbers in the ranges 0 to 9, 0 to 99 and 0 to 999. Here is a typical output:

```
RANDOM NUMBERS: RANGES
-------------------------------------------------
0 TO 9          0 TO 99         0 TO 999
-------------------------------------------------
 1               80              685
 3               50              174
 2               72              227
 4               9               649
 9               44              214
 7               3               570
 8               49              36
 9               65              20
 6               26              933
 8               73              888
-------------------------------------------------
```

The following program can be used to "check" the randomness of the Atari random number generator. It counts the number of 1s, 2s, 3s, 4s, 5s and 6s generated in a total list of 600. The counts should be approximately 100 each. Try running the program, it only takes a few seconds to provide the results.

```
10    REM *** RANDOM NO. CHECK ***
20    CLEARW 2:FULLW 2
30    RANDOMIZE 0
40    FOR N=1 TO 600
50    X=INT(RND*6+1)
60    IF X=1 THEN C1=C1+1
70    IF X=2 THEN C2=C2+1
80    IF X=3 THEN C3=C3+1
90    IF X=4 THEN C4=C4+1
100   IF X=5 THEN C5=C5+1
110   IF X=6 THEN C6=C6+1
120   NEXT N
130   PRINT "----------------------"
140   PRINT "NO. OF 1's ="; C1
150   PRINT "NO. OF 2's ="; C2
160   PRINT "NO. OF 3's ="; C3
170   PRINT "NO. OF 4's ="; C4
180   PRINT "NO. OF 5's ="; C5
190   PRINT "NO. OF 6's ="; C6
200   PRINT "----------------------------------"
```

Random numbers are extremely useful in testing programs, e.g. for sorting, statistics, etc. They are also widely employed in games programs. Here is a simple example:

```
10    REM *** Simple game using random nos. ***
20    clearw 2:fullw 2
30    print "This is a simple game, you versus
        the Atari"
40    print :print
50    print "Enter the number between 1 and 10"
60    input "you think the computer will
        generate";n
70    randomize 0
80    STnumber =int(rnd*11)
90    if n=STnumber then goto 130
100   print "Hard luck!"
110   print "The computer's no. was";STnumber
120   print:print:goto 140
```

```
130    print "Well,done, your guess was right"
140    print "If you wish another go type, yes"
150    print "if not , type no"
160    input ans$
170    if ans$="yes" then 50 else end
```

Trigonometric functions

Standard functions are available for the following trigonometric functions:

sin(x) gives the sine of x, where x is in radians

cos(x) gives the cosine of x, where x is in radians

tan(x) gives the tangent of x, where x is in radians,

atn(x) gives the inverse tangent, arctan x or $\tan^{-1}x$, with x given in radians in the range $-\pi/2$ to $+\pi/2$.

Note that in all cases x is assumed to be expressed in radians. The relationship between radians and degrees is:

x radians = x degrees $\times$ (π/180)

where x = 3.141593 (to six decimal places).

Figure 5.1 gives the basic definitions and the waveforms for sine, cosine and tangent.

Trigonometric functions: program examples

- The following program illustrates the use of the functions in determining sin x, cos x and tan x where x is input both in radians and degrees

```
10     rem *** Trig function example ***
20     clearw 2
30     input "Angle x in radians";x
40     print "sin(x) =";sin(x)
50     print "cos(x) =";cos(x)
60     print "tan(x) =";tan(x)
70     print "-----------------"
80     pi=3.141593
90     print x;" radians =";x*180/pi;"
         degrees2
100    input "Now enter angle in degrees";xd
```

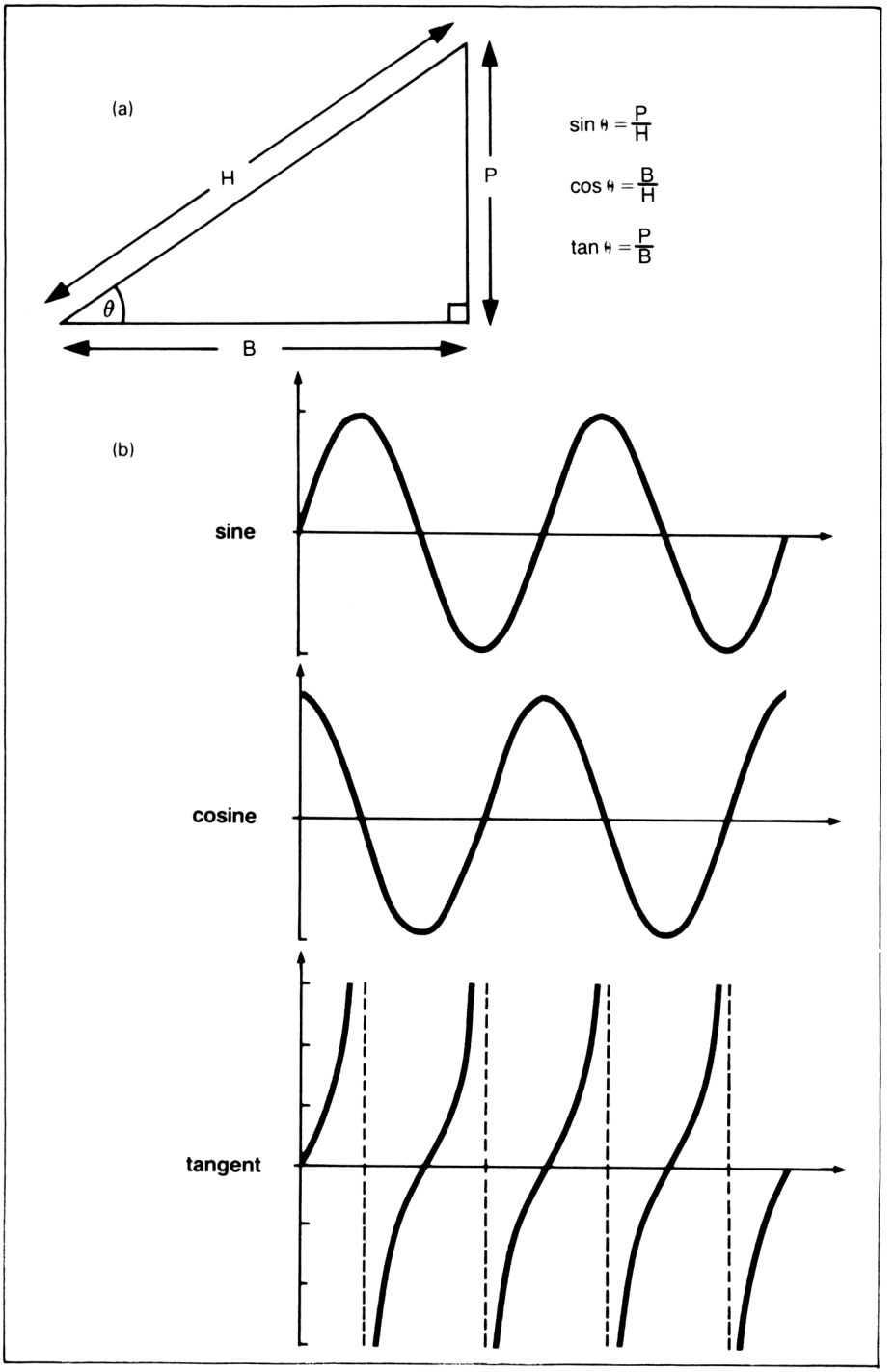


Figure 5.1 *The basic trigonometric functions:* (a) *definitions of sine, cosine and tangent of an angle;* (b) *waveforms for sine, cosine and tangent*

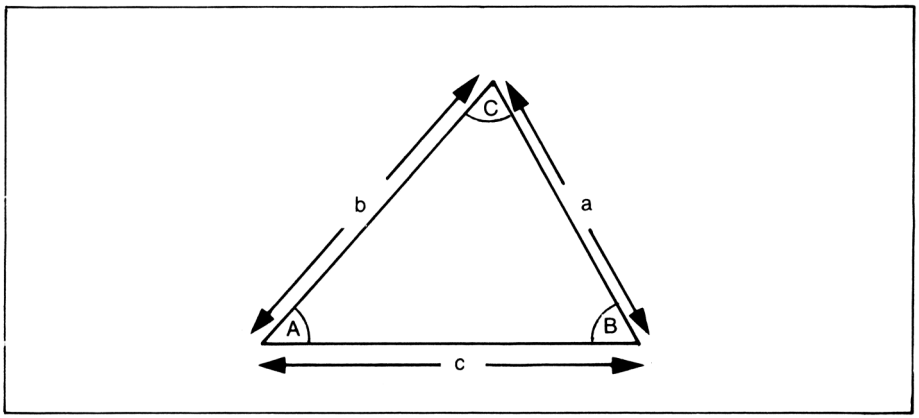


Figure 5.2 *Sine rule: a/sinA = b/sinB = c/sinC*

```
110    x=xd*pi/180
120    print "sine of ";xd;" degrees =";sin(x)
130    print "cosine =";cos(x)
140    print "tangent =";tan(x)
```

- This program applies the sine rule, see Figure 5.2, to calculate the sides BC and AB of a triangle given the angles A and C the length AC.

```
10     rem *** Sine rule example ***
20     clearw 2
30     input "Enter two known angles";A,C
40     input "Enter known length";AC
50     B=180-A-C
60     PI=3.141593
70     A=A*PI/180:B=B*PI/180:C=C*PI/180
80     BC=SIN(A)*AC/SIN(B)
90     AB=SIN(C)*AC/SIN(B)
100    PRINT "BC=";BC
110    PRINT "AB=";AB
```

Try running the program, for example, by inputting A = 32° , C = 75° and AC = 64.4. The results obtained for the two sides are: BC = 35.69, AB = 65.05.

- This program applies the second very useful rule, the cosine rule, see Figure 5.3:

```
10     REM *** APPLICATION OF COSINE RULE ***
20     CLEARW 2
30     INPUT "Two sides and included angle";
       b,c,A
```

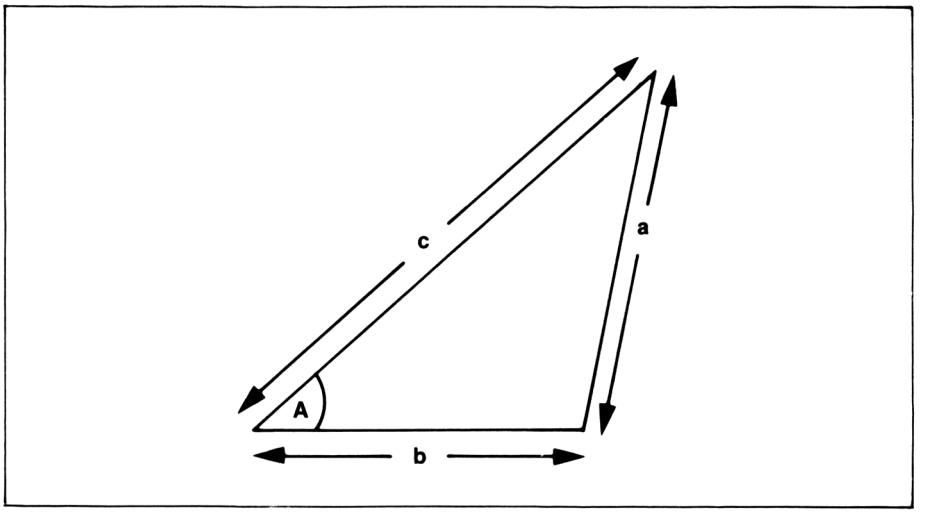


Figure 5.3 *Cosine rule:* $a^2 = b^2 + c^2 - 2bc\ cosA$

```
40      PI=3.141593
50      A=A*PI/180
60      BC=SQR(b^2+c^2-2*b*c*COS(A))
70      PRINT "Side BC ="BC
```

On running the program with the following data : b = 112, c = 58.6, A = 47°, you will obtain a = 83.8199.

Exponential and logarithmic functions

EXP(x) gives e raised to the power x (see Figure 5.4), i.e. gives values of the exponential function, e^x, where e = 2.7182818. Note: for the Atari, the value of x must lie within the range −43.6 to +43.6

LOG(x) gives the natural logarithm (log to base e) of x; where x must be greater than zero.

LOG10(x) gives the common logarithm (log to base 10) of x; x must be greater than zero.

Exponential and log functions: examples

1 PRINT EXP (2.4) [ENTER] or [RETURN]
11.0231764 value of $e^{2.4}$ obtained

2 PRINT EXP (1.6) + EXP (-1.6) [ENTER]
5.15492894 value of $e^{1.6} + e^{-1.6}$ displayed

3 The following program produces a table of $\log_e$ and $\log_{10}x$ for x = 1 to 10

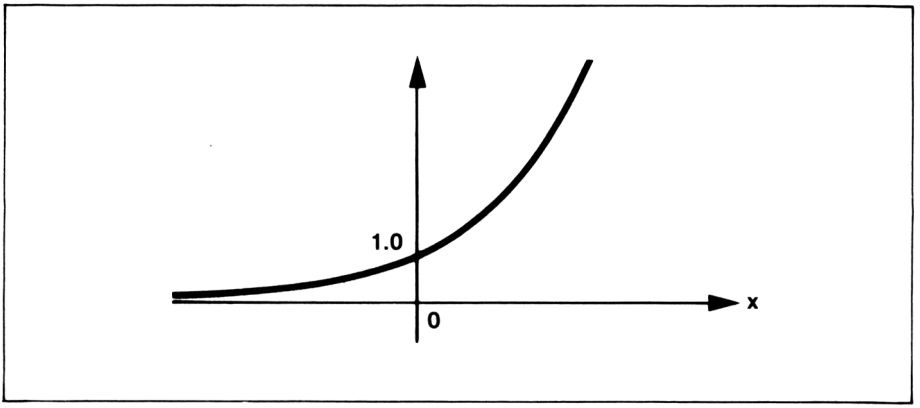


Figure 5.4 *Plot of exponential function* e^x

```
10      rem *** Natural and common logs ***
15      clearw 2:fullw 2
20      print "x","loge","log10"
30      print "-------------------------------------------"
40      for x=1 to 10
50      print x,log(x),log10(x)
60      next x
```

Here is the display:

```
x                loge             log10
-------------------------------------------
 1               0                0
 2               .693147          .30103
 3               1.09861          .477121
 4               1.38629          .60206
 5               1.60943          .698969
 6               1.79175          .778151
 7               1.9459           .845097
 8               2.07944          .90309
 9               2.19722          .954242
 10              2.30258          1
```

User-defined functions

The BASIC language also allows us to define our own functions, user-defined functions, as follows:

100 DEF FN identifier (parameter list) = expression for result

The function must have an identifier, just as a variable, and the parameter is used to feed data values to be used in computing the function result, that is, the value returned by the function.

A user-defined function can be "called" or "invoked" at any time in a program simply by writing FN identifier (parameter list values).

The function definition, however, should be written outside any loops in which it is called and certainly always before it is first used. Function definitions are therefore usually written early on in a program.

The definition and function "calls" are illustrated in the following examples.

User-defined functions: program examples

- This program contains the definitions of two functions:

  FNA(r) = $4\pi r^2$ and FNV(r) = $4\pi r^3/3$

  to calculate, respectively, the surface area and the volume of a sphere of radius r.

  Calls are made to the functions: at line 60 to calculate the area for r = 56.9; at line 70 to calculate the volume for r also = 56.9; and at line 100 to calculate area and volume for a value of r input from the keyboard.

```
10    rem * User defined function example *
15    clearw 2
20    print "Surface area and volume of sphere"
30    def fnarea(r)=4*3.14159*r^2
40    def fnvolume(r)=4*3.14159*r^3/3
50    print "Surface area for r=56.9 is;"
60    print fnarea(56.9)
70    print "Volume =";fnvolume(56.9)
80    print "-------------------------"
90    input "Enter radius";r
100   print "for r=";r
110   print "Area =";fnarea(r)
120   print "Volume =";fnvolume(r)
```

- This program contains the function,

  fni (p,r,t)

  which is used to calculate compound interest.

The program calls the function in a FOR loop to calculate a table of interest values for t = 1 to 10 years; the values for the sum invested p and interest rate are input from the keyboard.

```
10      rem *** User defined function ***
15      rem  ** for interest calcs, **
20      clearw 2
30      def fni(p,r,t)=p*(1+r/100)^t-p
40      input "Sum invested and rate %";p,r
50      print "year","interest"
60      print "-----------------------------"
70      for t=1 to 10
80      print t,fni(p,r,t)
90      next t
```

The display obtained when inputting p = 1000, r = 15 is

```
year              interest
-----------------------------
 1                 150
 2                 322.499
 3                 520.874
 4                 749.005
 5                 1011.35
 6                 1313.05
 7                 1660.01
 8                 2059.01
 9                 2517.87
 10                3045.55
```

Strings and string slicing

BASIC also provides several useful functions for processing strings. In this section we consider string "slicing" functions, that is, functions that can remove characters from a string to form a smaller string.

The LEFT$ function

The LEFT$ function is used, as its name suggests, to obtain the first N characters of a string starting from the left. A LEFT$ instruction takes the form:

LEFT$(A$,N)

which would produce a string consisting of the first N characters of A$. Its action is illustrated by running the following programs:

```
10      rem *** left$ demo ***
15      clearw 2
20      a$="123456789"
30      b$="abcdefghijklmn"
40      print "first 4 characters from left "
50      print "of a$ is ";left$(a$,4)
60      print
70      print "first 8 characters from left"
80      print "of b$ is ";left$(b$,8)
```

The display obtained in the output window is:

```
first 4 characters from left
of a$ is 1234

first 8 characters from left
of b$ is abcdefgh
```

The RIGHT$ function

The RIGHT$ function is used to obtain the last N characters of a string, ending with the rightmost character. For example, the display obtained when running the following program:

```
10      rem *** right$ demo ***
20      print right$("Meadows R G",3)
30      print
40      a$="ABCDEFGHIJ"
50      print right$(a$,6)
```

is

```
R G          last 3 characters of "Meadows R G"
EFGHIJ       last 6 characters of a$
```

The MID$ function

The MID$ function allows any portion of a string to be obtained. It takes the general form,

MID$(A$,M,N)

where the whole numbers (integers) M and N define that the string so formed

consists of N characters starting from the Mth character from the left. For example,

```
10      rem *** mid$ demo ***
20      print mid$("Meadows R G",4,2)
30      print
40      a$="ABCDEFGHIJ"
50      print mid$(a$,5,3)
```

produced the output display

```
do          2 characters, starting 4th from left
EFG         3 characters, starting 5th from left
```

The LEN function

The LEN standard function gives the length, i.e. the number of characters, in a string. For example,

- PRINT LEN "How many characters in this string?" [RETURN]
  35 . . .result displayed.

  Remember all characters including spaces, punctuation marks, etc., in the string count as part of its "length".

- ```
  10      rem *** len demo ***
  20      a$="R G Meadows"
  30      print len(a$)
  40      print
  50      print len("123456789abcdef")
  ```

  RUN [RETURN]
  11 . . .result for LEN(a$)
  15 . . .result for LEN ("123456789abcdef")

VAL and STR$ functions

The VALue function converts a "number" string into the actual number which can then be assigned, for example, to a numeric variable.

The STR$ function is the inverse; it converts numeric data into a string.

The following short program demonstrates their action.

```
10      rem *** VAL and STR$ demo ***
20      a$="4453"
```
```

```
30      n=val(a$)
40      print a$,n,n*n
50      print
60      b$=str$(n)+str$(n)
70      print b$
```

RUN [RETURN}

```
4453            4453            1.98292E+07

 4453 4453
```

The first line of output (line 40) displays

a$ (i.e. "4453") n (i.e. 4453) and n × n.

Clearly the VAL function has converted the string "4453" in the number 4453.

The second line of output represents the string b$ formed by combining the strings "4453" and "4453".

Character sets and codes

The characters, i.e. letters, digits 0, 1, 2,..., punctuation marks and arithmetic symbols that appear on the Atari keyboard form part of what is known as the character set of the computers.

Each character is assigned a number code. The function CHR$(N) converts this code number N to its character equivalent in the character set of the Atari ST computer. In fact the code is an internationally recognised one known as the ASCII code. See also below for the inverse ASC function.

The following program displays a portion of the ASCII character set from n = 33 to 126. This range covers most of the keyboard characters.

```
10      rem *** ASCII character set ***
20      clearw 2
30      print "ASCII ","Character"
40      print "code no."CHR$(n)"
50      print"--------------------------"
60      for n=33 to 126
70      print n,chr$(n)
80      next n
```

You will obtain a line-by-line display if you run the program. Figure 5.5 shows part of the ASCII character set.

Remember you can use BREAK to pause the program execution (and hence also the display) and press the RETURN key or type in CONT to continue.

The ASCII code is one of the most commonly used codes for transmitting information between computers and associated devices, such as printers, and is the abbreviation for American Standard Code for Information Interchange.

The inverse process of obtaining the number equivalent of the ASCII code of a character can be found using the ASC("X") function.

The ASC function returns the decimal number code for the character X within the quotation marks or, if a string is present, the first character of the string. Obviously the character must be a member of the ASCII set. For alphabetic characters, digits, punctuation marks, etc., the ST Atari code is identical to the ASCII codes.

Character	Code	Character	Code	Character	Code
0	48	A	65	a	97
1	49	B	66	b	98
2	50	C	67	c	99
3	51	D	68	d	100
4	52	E	69	e	101
5	53	F	70	f	102
6	54	G	71	g	103
7	55	H	72	h	104
8	56	I	73	i	105
9	57	J	74	j	106
:	58	K	75	k	107
;	59	L	76	l	108
<	60	M	77	m	109
=	61	N	78	n	110
>	62	O	79	o	111
?	63	P	80	p	112
space	32	Q	81	q	113
!	33	R	82	r	114
"	34	S	83	s	115
#	35	T	84	t	116
$	36	U	85	u	117
%	37	V	86	v	118
&	38	W	87	w	119
'	39	X	88	x	120
(	40	Y	89	y	121
)	41	Z	90	z	122

Figure 5.5 *Part of ASCII code character set obtained by running ASCII character set – CHR$(n) program*

For example,

- PRINT ASC("A") [RETURN]
  65
  ASCII code for A

- PRINT ASC("a") [RETURN]
  97
  ASCII code for a

- PRINT ASC("+") [RETURN]
  43
  code for + symbol

- PRINT ASC("676") [RETURN]
  54
  code for 6, the first character of the string

- PRINT ASC ("String here") [RETURN]
  83
  code for S

Denary-to-hexadecimal conversion

The hex$ function can be used to convert denary (normal base 10 numbers) to hexadecimal representation. Hexadecimal numbers are base 16 numbers and are often used in computing to identify the addresses of memory locations.

The following program displays a table of decimal numbers and the corresponding hexadecimal representation:

```
10      rem *** Hexadecimal nos. ***
20      clearw 2
30      print "Decimal no.","Hexadecimal"
40      print "--------------------------"
50      for n%=1 to 16
60      print n%,hex$(n%)
70      next n%
```

The display obtained is:

Decimal no.	Hexadecimal
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10

EXERCISE PROBLEMS 5.1

1 Use the functions CINT, INT and FIX appropriately to
   (a) Round the following to the nearest whole number:
   4.69, 587.3, −199.9, 0.678, −0.989
   (b) To specify the following to an accuracy of two decimal places:
   58.632998, − 9.9989, 0.46798, √58.67.

2 Write a program to display a list of 16 random numbers in the range 1 to 55. Use it to generate a treble-chance forecast!

3 Using the cosine rule: $a^2 = b^2 + c^2 - 2bc \cos A$ write programs,
   (a) to find the angles of a triangle when given the lengths a,b,c
   (b) to determine a and angles B and C when given lengths b and c and angle A.

4 Write functions that will return the following values
   (a) the average of 3 quantities
   (b) the volume of a cylinder, given $V = \pi r^2 h$ where r = radius and h = height of cylinder.

5 Use the LEN function to determine the lengths of the strings:
   (a) "42 Lynton Avenue, London NW55"
   (b) A$ = "A stitch in time, save 9"

(c) A$ + B$, where A$ = "123456789" and B$ = "abcdefghihklmnopq".

6 Find
(a) the ASCII code for Z, ; 9 − = * !
(b) the characters corresponding to ASCII codes 118, 35, 59, 86.

CHAPTER 6

APPLICATIONS OF ARRAYS AND FILES

Introduction and summary

The subject of arrays and files is very often considered as rather advanced and really the domain of the more experienced programmer. However, with the Atari this is really not so – arrays which are essentially easy-to-access storage "boxes" for the values we wish to enter, use and process in our programs and files, which allow us to keep permanent records of input data, results, etc. – are very easy to use. They provide much greater scope for writing more interesting programs and allow us to extend our activities to a very much broader field of applications.

In this chapter we first explain the meaning of arrays and explain how they may be generated using the DIM statement. We then consider how they may be used: how data may be fed in, processed and output. We consider also application to sorting numerical and alphabetic information in order. Finally the idea of a file for keeping permanent records is introduced and we explain how a file can be created and accessed on disk.

Arrays and the DIM statement

So far we have used individual variables in our programs. Each variable we used was given a name – its identifier – and this identifier was used to identify a storage location, a "box" or "container", to store the data assigned to the variable at a convenient place in

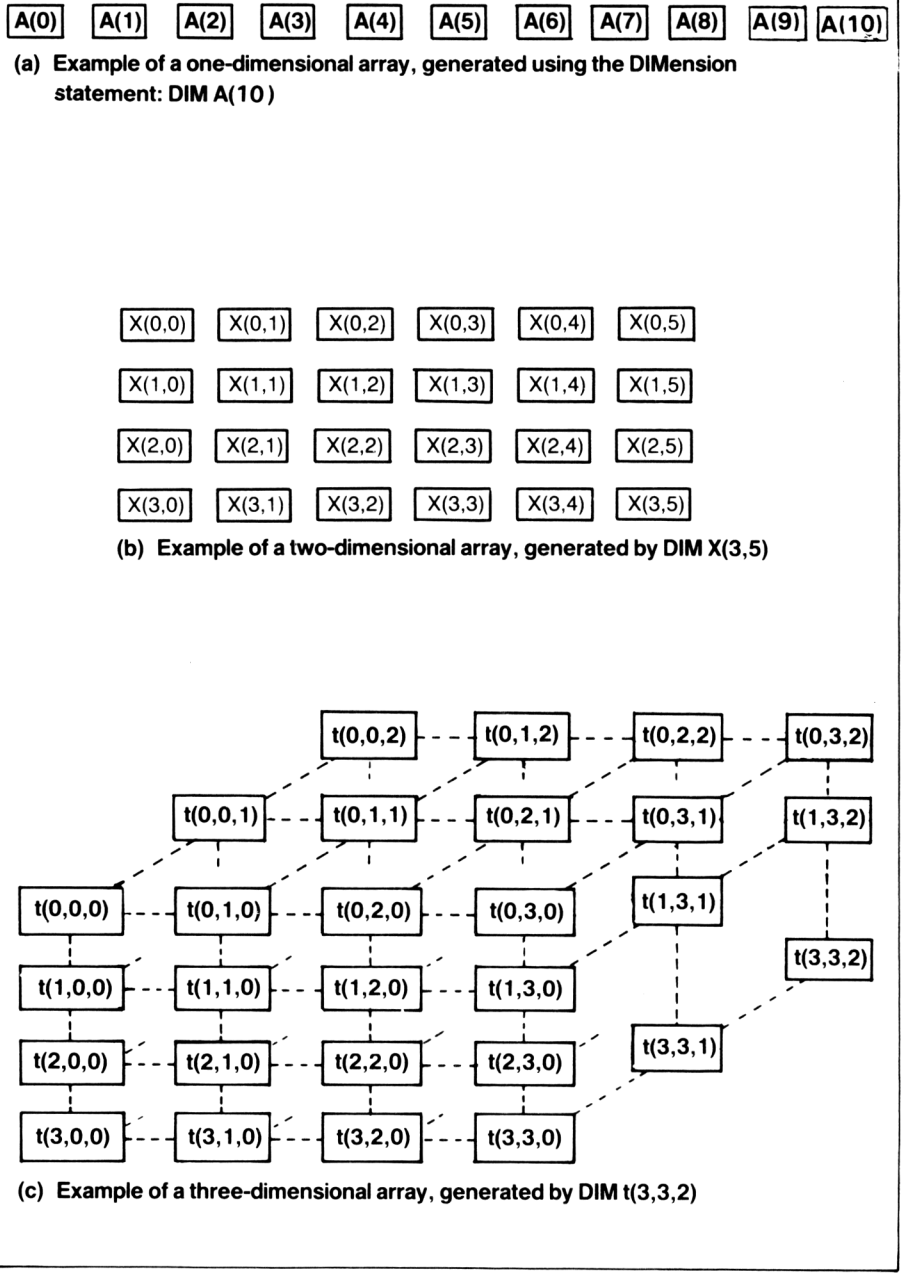

A(0) A(1) A(2) A(3) A(4) A(5) A(6) A(7) A(8) A(9) A(10)
(a) Example of a one-dimensional array, generated using the DIMension statement: DIM A(10)
X(0,0) X(0,1) X(0,2) X(0,3) X(0,4) X(0,5)
X(1,0) X(1,1) X(1,2) X(1,3) X(1,4) X(1,5)
X(2,0) X(2,1) X(2,2) X(2,3) X(2,4) X(2,5)
X(3,0) X(3,1) X(3,2) X(3,3) X(3,4) X(3,5)
(b) Example of a two-dimensional array, generated by DIM X(3,5)
t(0,0,2) t(0,1,2) t(0,2,2) t(0,3,2)
t(0,0,1) t(0,1,1) t(0,2,1) t(0,3,1) t(1,3,2)
t(0,0,0) t(0,1,0) t(0,2,0) t(0,3,0) t(1,3,1)
t(1,0,0) t(1,1,0) t(1,2,0) t(1,3,0) t(3,3,2)
t(2,0,0) t(2,1,0) t(2,2,0) t(2,3,0) t(3,3,1)
t(3,0,0) t(3,1,0) t(3,2,0) t(3,3,0)
(c) Example of a three-dimensional array, generated by DIM t(3,3,2)

Figure 6.1 *Pictorial representation of arrays: each "box" in the array can be considered as an individual storage location for an array element variable*

the computer's memory. An array is used in the same way and allows us to declare a whole number of storage locations using a single DIMension statement.

Figure 6.1 provides a pictorial representation which should help you to understand the concept of an array as a means of storing not one but several different items of data. Every element in the array can be used to store data and can be individually accessed.

Each array we use in a program is given its own general identifier, just as we assign names or identifiers to variables. Thus we can give for an associated group of variables a collective name – the array identifier – rather than a whole list of different identifiers. We can easily refer to an individual element, i.e. storage location or "box", by the array identifier plus number subscript(s), the number(s) specified within the brackets in Figure 6.1.

The use of arrays saves a great amount of writing space in preparing programs and is very useful for inputting, processing and outputting large volumes of information.

Arrays are created in programs using the DIMension statement:

100 DIM array identifier (N) . . .for one-dimensional arrays
200 DIM array identifier (N1, N2) . . .for two-dimensional arrays
300 DIM array identifier (N1, N2, N3) . . .for three-dimensional arrays

The numbers N, N1, N2, N3 within the brackets specify the overall size of DIMensions of the array, i.e. the total number of elements (storage locations for variable values) in the array. When the computer "sees" a DIM statement it puts aside memory space for the array elements as specified by the numbers within the brackets.

Arrays can be defined for numeric data (real and integer) and for strings. The array identifiers for integer data must end with the % symbol and for strings with the $ sign, just as for "normal" variables.

Examples of the DIM statement

One-dimensional arrays The statement,

100 DIM A(10)

defines a one-dimensional array. The array identifier is A and the "dimension" of the array is 10. A dimension of 10 normally creates 10 + 1 = 11 elements, see Figure 6.1(a). Each element is identified by the array identifier and a subscript or index enclosed within brackets. The subscripts run from 0 to the number specified in the DIM statement. Thus the above statement creates the following 11 elements:

x(A),A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8),A(9) and A(10)

for numerical data.

If you wish to set the minimum value of the array subscript to 1 rather than 0, use the OPTION BASE 1 statement, e.g. the statements

```
90 OPTION BASE 1
100 DIM A(10)
```

generate 10 elements, with the first element subscripts commencing at 1, i.e. the first element would now be A(1).

If no OPTION BASE statement is used, the array subscripts always begin at 0 with the lowest element then being A(0). If you have previously incorporated an OPTION BASE 1 statement, you can reset to the 0 subscript by using the statement

```
110 OPTION BASE 0
```

Thus a subsequent statement

```
120 DIM item$ (2))
```

would create an array for string data, identifier item$, consisting of 20 + 1 elements.

Two-dimensional arrays The statement, assuming OPTION BASE 0,

```
130 DIM X (3,5)
```

creates a two-dimensional array for numeric (real) variable data consisting of a total of (3 + 1) (5 + 1) = 24 elements, see Figure 6.1(b). In this case each element is identified by a pair of subscripts, the first of which can be thought as specifying the row and the second as specifying the column of the element in the array.

Three-dimensional arrays The statement, again assuming

```
140 DIM t (3,3,2)
```

creates a three-dimensional array, identifier in this case t, which consists of a total of (3 + 1) × (3 + 1) × (2 + 1) = 48 elements for real variable data, see Figure 6.1(c).

Use and application of arrays

Basically arrays are used, just like individual variables, for storing and processing information. Values for the array elements can be input from the keyboard using the INPUT statement or READ in from DATA statements within the program.

The following examples illustrate some basics concerning the handling and application of arrays.

- This example illustrates how data can be READ into the array elements and how the stored data in the elements may be accessed and displayed:

```
10      rem *** Array demo 1 ***
20      rem ** Reading data into an array **
30      clearw 2
40      dim n(10)
50      for i=0 to 10
60      read n(i)
70      next i
80      rem *** displaying array contents ***
90      lprint "item no.","  n"
100     for i=1 to 10
110     print i,n(i)
120     next i
130     data 76,92,34,71,64,39
140     data 121,63,85,49,110
```

Line 40 defines a 10 + 1 element array. The FOR loop, lines 50 to 70, READS in data to each of the 11 array elements. The display of this information is produced by the FOR loop, lines 100 to 120. Thus on running the program the following list will be displayed; note by commencing at i = 1 the x(0) item is not displayed in this case.

```
item no.          n
 1               92
 2               34
 3               71
 4               64
 5               39
 6              121
 7               63
 8               85
 9               49
 10             110
```

- This second example illustrates the handling of both numeric and string arrays, defined respectively at lines 40 and 50. The corresponding data is contained in the data statement lines 150 to 180. Note: do not enclose the string data within quotation marks. Not only is it unnecessary but will lead to your program not running. Once again FOR loops are used to READ in and PRINT out the array element values.

```
10      rem *** Array demo 2 ***
20      rem ** Reading data into 2 arrays **
30      clearw 2:fullw 2
40      dim n(10)
50      dim t$(10)
60      for i=0 to 10
70      read n(i),t$(i)
80      next i
90      rem *** displaying contents of both
          arrays **
100     lprint "item no.","number","name"
110     lprint "-------------------------------"
120     for i=1 to 10
130     lprint i,n(i),t$(i)
140     next i
150     data 76,lemons,23,melons,112,bananas
160     data 121,pears,49,apples,55,grapes
170     data 432,tangerines,31,pineapples
180     data 52,cherries,62,apricots,11,limes
```

which displays

```
item no.        number          name
-------------------------------------
 1               23             melons
 2               112            bananas
 3               121            pears
 4               49             apples
 5               55             grapes
 6               432            tangerines
 7               31             pineapples
 8               52             cherries
 9               62             apricots
 10              11             limes
```

- This program extends the scope of the last two by dealing with three arrays: n for number of items, t$ for item description and p for the unit price of the item. The three arrays are defined in lines 30, 40 and 50. A FOR loop is used to READ in the respective values which are contained in the DATA statements 220 to 260.

  A FOR loop is also used, see lines 140 to 170, to display the list in tabular form and at the same time calculate sub-totals and total value of the stock.

```
10    rem *** Array demo 3 ***
20    clearw 2:fullw 2
30    dim n(10)
40    dim t$(10)
50    dim p(10)
60    option base 1
70    for i=1 to 10
80    read n(i),t$(i),p(i)
90    next i
100   rem *** displaying array contents ***
110   rem **  and calculating sub-totals **
120   print "item ","number","unit-price",
        "sub-total"
130   print "-------------------------------"
140   for i=1 to 10
150   print t$(i),n(i),p(i),p(i)*n(i)
160   sum=sum+p(i)*n(i)
170   next i
180   print "-------------------------------"
190   print "Total stock value =";
200   print using "#######.##";sum
210   print "-------------------------------"
220   data 76,1.2L saloon,4350,23,2.2L
        basic,4995
230   data 45,1.6L hatch,5150,79,mini,2999
240   data 42,sports,6950,30,2L estate,6500
250   data 34,XL spec,8450,2,RR,85000
260   data 12,4.4L XJ,18900,5,Range R,12380
```

On running the program you will obtain the following display in the output window:

```
item            number  unit-price     sub-total
-------------------------------------------------
1.2L saloon       76       4350.12        330609
2.2L basic        23       4995           114885
1.6L hatch        45       5150           231750
mini              79       2999           236921
sports            42       6950           291900
2L estate         30       6500           195000
XL spec           34       8450           287300
RR                2        85000          170000
4.4L XJ           12       18900          226800
Range R           5        12380          61900
-------------------------------------------------
Total stock value =2147056.00
-------------------------------------------------
```

- The following example illustrates the READing in and display of numerical data using a two-dimensional array. Once again we use FOR loops but in this case FOR loop within another one. The "i" loop sets the row while the "j" loop runs across the row reading the various columns. Try running the program.

```
10    REM * TWO DIMENSIONAL ARRAY EXAMPLE *
20    clearw 2
30    dim r(5,4)
40    rem ** Reading in data **
50    for i=1 to 5
60    for j=1 to 4
70    read r(i,j)
80    next j
90    next i
100   rem ** Display of array elements **
110   print "element","value"
120   for i= 1 to 5
130   for j= 1 to 4
140   print i;j,r(i,j)
150   next j
160   print
170   next i
180   data 23,65,78,21,90
190   data 84,39,72,81,32
200   data 71,64,73,29,63
210   data 88,40,21,63,80
```

Here is the output giving both the subscript values as well as the array element values:

```
element          value
 1   1            23
 1   2            65
 1   3            78
 1   4            21
 2   1            90
 2   2            84
 2   3            39
 2   4            72
 3   1            81
 3   2            32
 3   3            71
 3   4            64
 4   1            73
 4   2            29
 4   3            63
 4   4            88
 5   1            40
 5   2            21
 5   3            63
 5   4            80
```

- Here is an example of the use of arrays with strings: a simple French-English vocabulary test.

  Arrays are used to store the English and corresponding French words. On running the program you are asked to type in the English word for a given French one. The computer tells you if you are right and if you are wrong gives the correct English word.

  Try running the program and then perhaps extend it for more words.

```
10      rem *** ENGLISH - FRENCH TEST ***
20      rem ** Reading in data **
30      option base 1
40      dim English$(8)
50      dim French$(8)
60      print "Words current stored"
70      for n=1 to 8
80      read English$(n),French$(n)
```

```
90    print English$(n),French$(n)
100   next n
110   rem * Short delay for observation
120   for d=1 to 2000:next d
130   clearw 2
140   n=1
150   print "Test begins"
160   print "Enter English word for ";
        French$(n);
170   input answer$
180   if answer$=English$(n) then 190
        else 240
190   print "Well done, correct
200   n=n+1
210   print
220   if n>8 then ? "Test completed":end
230   goto 160
240   print "sorry you are wrong"
250   print "The answer is  ";English$(n)
260   n=n+1
270   print
280   if n>8 then ? "Test completed":end
290   goto 160
300   data road,rue,child,enfant,wine,vin
310   data sea,mer,door,porte,ten,dix
320   data bread,pain,tree,arbre
```

Sorting numbers in order

An important and very interesting application for which the computer is ideally suited is in the sorting of data in order.

The following program segment may be used to sort numerical data in ascending order:

```
200   rem *** Bubble sort routine ***
210   rem n=no. of terms to be sorted
220   for k=1 to n-1
230   for l=k+1 to n
240   if x(l)<x(k) then swap x(l),x(k)
250   next l
260   next k
270   rem *** end of sort routine ***
```

The data to be sorted is first fed into an array x(n) consisting of elements x(1), x(2), . . ., up to x(n). The above routine is then brought into action. It starts by comparing the first data item x(1) with its neighbour x(2) and if the first item is greater then the two values are swapped in position in the array using the ST BASIC standard function SWAP. The comparison-swap action then continues with first and third elements. . .and so on until the minimum value heads the list, i.e. x(1) stores the minimum value. The routine is then repeated with the second item in the list, and so on, to the last term, so at the end of the second sort sequence the first two terms are in order. Execution then moves onto the third term, etc., and the whole comparison-swap action is repeated until we finally end up with all elements in ascending order. A pictorial representation for the simple case of sorting five values in ascending order is illustrated in Figure 6.2.

The "bubble-sort" routine, so called because the data "bubbles" up or down in the sort may be easily changed to sort in descending order. Simply change the "less than" symbol in line 240 to the "greater than" symbol >, i.e. 240 IF x(1) > THEN SWAP x(l), x(k).

The following program illustrates the application of the bubble-sort method to sort 50 numbers in ascending order. The 50 numbers are contained in the data statements 280 to 320.

```
10    rem *** Sorting numbers in order ***
20    rem * Reading in data *
30    option base 1
40    dim x(50)
50    clearw 2:fullw 2
60    print "Unsorted list"
70    for n= 1 to 50
80    read x(n)
90    print x(n);
100   next n
110   print "--------------------------------"
120   gosub 200:rem ** to sor routine **
130   print:print
140   print "Sorted list; ascending order"
150   for n=1 to 50
160   print x(n);
170   next n
180   print "--------------------------------"
190   end
200   rem *** Bubble sort routine ***
210   n=50
220   for k=1 to n-1
```

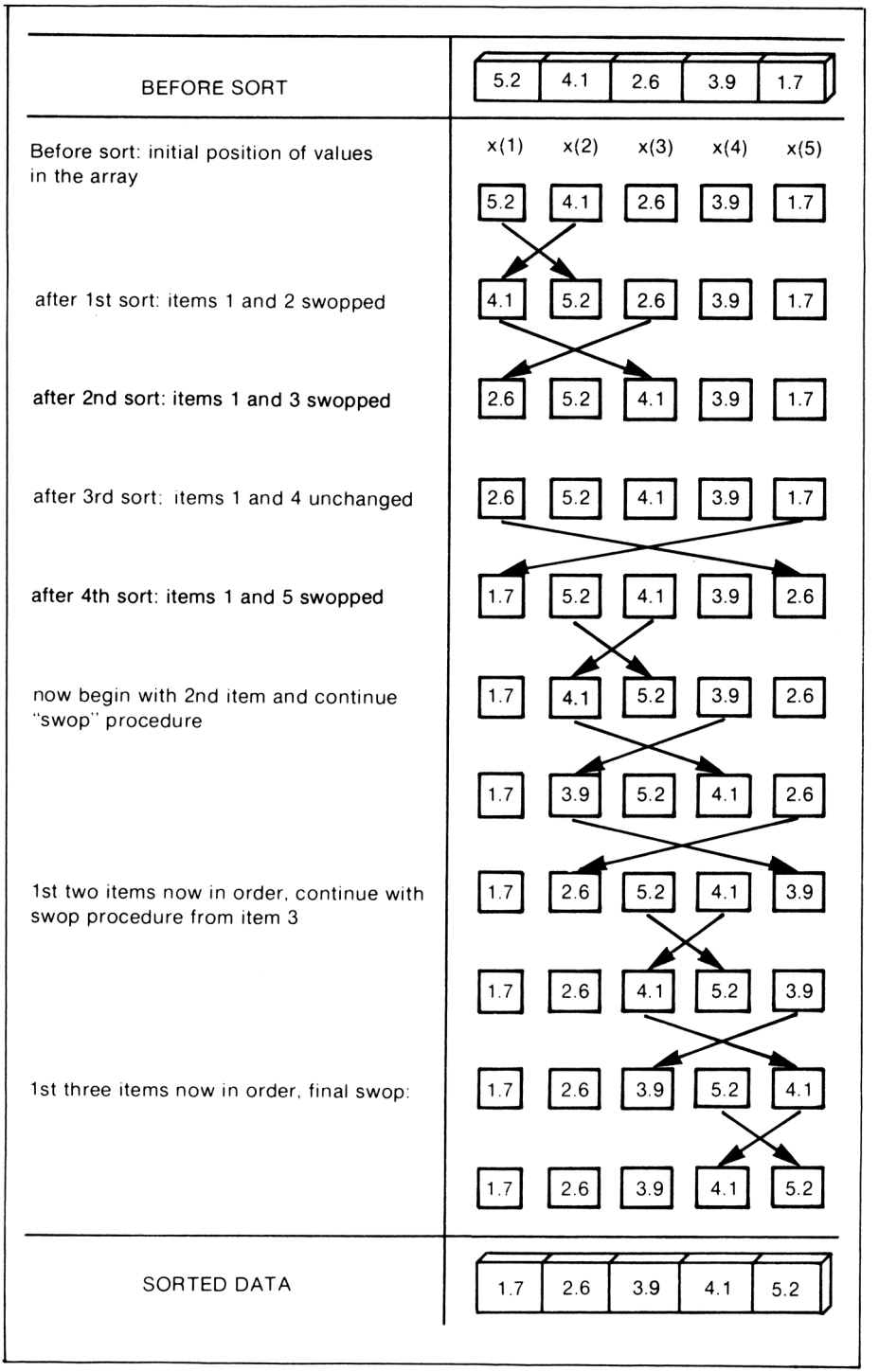

BEFORE SORT
5.2 4.1 2.6 3.9 1.7
Before sort: initial position of values in the array
x(1) x(2) x(3) x(4) x(5)
5.2 4.1 2.6 3.9 1.7
after 1st sort: items 1 and 2 swopped
4.1 5.2 2.6 3.9 1.7
after 2nd sort: items 1 and 3 swopped
2.6 5.2 4.1 3.9 1.7
after 3rd sort: items 1 and 4 unchanged
2.6 5.2 4.1 3.9 1.7
after 4th sort: items 1 and 5 swopped
1.7 5.2 4.1 3.9 2.6
now begin with 2nd item and continue "swop" procedure
1.7 4.1 5.2 3.9 2.6
1.7 3.9 5.2 4.1 2.6
1st two items now in order, continue with swop procedure from item 3
1.7 2.6 5.2 4.1 3.9
1.7 2.6 4.1 5.2 3.9
1st three items now in order, final swop:
1.7 2.6 3.9 5.2 4.1
1.7 2.6 3.9 4.1 5.2
SORTED DATA
1.7 2.6 3.9 4.1 5.2

Figure 6.2 *Pictorial representation of bubble-sort method*

```
230    for l=k+1 to n
240    if x(l)<x(k) then swap x(l),x(k)
250    next l
260    next k:rem *** end of sort routine **
270    return
280    data 12,34,61,90,54,61,67,92,81,11
290    data 55,23,76,41,59,61,23,45,19,65
300    data 39,21,14,68,76,55,44,89,23,17
310    data 66,48,81,34,69,49,23,26,12,64
320    data 14,45,69,98,65,32,12,34,74,67
```

On running the program you will obtain immediately the list of unsorted numbers and then within a few seconds the list sorted in ascending order. Here is the actual display:

```
Unsorted list
 12   34   61   90   54   61   67   92   81   11   55   23
 76   41   59   61   23   45   19   65   39   21   14   68
 76   55   44   89   23   17   66   48   81   34   69   49
 23   26   12   64   14   45   69   98   65   32   12   34
 74   67
Sorted list; ascending order
 11   12   12   12   14   14   17   19   21   23   23   23
 23   26   32   34   34   34   39   41   44   45   45   48
 49   54   55   55   59   61   61   61   64   65   65   66
 67   67   68   69   69   74   76   76   81   81   89   90
 92   98
```

Sorting in alphabetical order

It is a straightforward matter to adapt a numerical sort routine to sort string variable data, e.g. names, into alphabetic order. Firstly the arrays to hold the data must obviously be defined as string variable arrays so include the $ sign in the array identifier. Secondly use the ASC standard function to convert the first letter of the string variable value to its number code.

By doing this, number comparison may be made. Remember: ASC(X$(l)) returns the ASCII code for the first character of the string variable x$(l) value; and that these codes are ordered and cover the ranges,

65, 66,. . .90	for A,B, . . .Z
97, 98,. . .122	for a,b,. . .z

Thus modifying the numeric bubble-sort routine to a form for sorting string data in alphabetic order leads to:

```
200    rem *** Bubble sort routine ***
210    rem n=no. of terms to be sorted
220    for k=1 to n-1
230    for l=k+1 to n
240    if ASC(x$(l))<ASC(x$(k)) then swap
          x$(l),x$(k)
250    next l
260    next k
270    rem *** end of sort routine ***
```

The following program illustrates it use. Array x$(30) is used to store up to 30 names, each name being stored in one of the array elements x$(1), x$(2), ..., x$(30). Since there are only 10 names in our example only 10 elements are required.

```
10     rem *** Sorting names in order ***
20     rem * Reading in data *
30     option base 1
40     dim x$(30)
50     clearw 2:fullw 2
60     print "Unsorted list"
70     print "-------------"
80     for n= 1 to 30
90     read x$(n)
100    print x$(n);"  ";
110    next n
120    print
130    gosub 220:rem ** to sort routine **
140    print:print
150    print "Sorted list: alphabetic order"
160    print "-----------------------------------"
170    for n=1 to 30
180    print x$(n);"  ";
190    next n
200    print "--------------------------"
210    end
220    rem *** Bubble sort routine ***
230    n=30
240    for k=1 to n-1
250    for l=k+1 to n
260    if ASC(x$(l))<ASC(x$(k)) then swap
         x$(l),x$(k)
270    next l
```

```
280    next k:rem *** end of sort routine **
290    return
300    data VENN,GAUSS,BROWM,FOSS,WINN,HARD,SMITH
310    data CHARLTON,DUNN,HARRIS,ISSACS,CRAMM
320    data TALBOT,RUSSEL,WARWICK,LEWIN,PARK
330    data GROSS,JACKS,LIST,TOWNSEND,EDWARDS
340    data APPLETON,KING,MEAD,NUTT,QUEEN
350    data POWELL,URWIN,ZEBERDEE
```

Here is the output obtained on running the program.

```
Unsorted list
-------------
VENN  GAUSS  BROWM  FOSS  WINN  HARD  SMITH  CHARLTON  DUNN  HARRIS
ISSACS  CRAMM  TALBOT  RUSSEL  WARWICK  LEWIN  PARK  GROSS  JACKS  LIST
  TOWNSEND  EDWARDS  APPLETON  KING  MEAD  NUTT  QUEEN  POWELL  URWIN
ZEBERDEE


Sorted list: alphabetic order
-----------------------------
APPLETON  BROWM  CRAMM  CHARLTON  DUNN  EDWARDS  FOSS  GAUSS  GROSS
HARD  HARRIS  ISSACS  JACKS  KING  LIST  LEWIN  MEAD  NUTT  PARK  POWELL
  QUEEN  RUSSEL  SMITH  TOWNSEND  TALBOT  URWIN  VENN  WINN  WARWICK
ZEBERDEE
```

Note that the program does not sort effectively names beginning with the same letter. Modify the program using, for example, the MID$ function and further comparison statements to solve this problem.

Using files in programs

In all our programming work so far we have input data to be processed by our program either directly from the keyboard or by using READ statements and outputting results to the output window for display or to a printer for a hard copy. As soon as the program has been executed all keyboard data entered and any intermediate results are effectively lost.

How can we store, for example, input data on a permanent basis and feed it directly into our program when required, rather than using the keyboard or DATA statement as the source? How can we store data obtained from one part of a program for processing in another (without, of course, using arrays, which, although ideally suited to this task, may take up excessive and valuable memory space in the computer)? How can we store our output data for use in another program?

These problems may be overcome by storing input, output and intermediate

"transferable" data in files in an auxiliary memory storage device such as a magnetic disk. We can then prepare and input data independently, save it on disk and read in the data when required in program execution. Likewise if we wish to save the output results on a permanent basis we can direct these to be written to disk.

The Atari ST computers handle both sequential and random access files. A sequential file can only be examined one element at a time, starting from the beginning and then progressing through in the order in which they were originally stored. Unlike an array, whose individual elements can be directly or "randomly" accessed, data in a sequential file cannot. File size, however, is not specified and a file may be allowed to expand, within the limits of available memory storage space, to any size. The Atari ST computers also provide random access files, which, like arrays, can be accessed at random.

With these introductory comments in mind let us see how we can create, write to and read from files on disk.

Sequential files
To open a file on disk and to write data output to the disk file:

- Whether we are writing to or reading from a disk file we must first open the file. To open a sequential file for writing to use:

  10 OPEN "O", N, "filename"

  the "O" indicates Output; you are to output data to be stored on the disk.

  N is the file channel number; value to be selected between 1 and 15 (any number in this range can be selected and this number identifies the channel of communication between the computer and the file on disk; it remains allocated to the particular file so long as the file remains OPEN); "filename". . .the name under which you wish your file to be stored.

- To write data to the file use statements of the form:

  20 PRINT # N, "strings"
  30 PRINT # N, x,y,z

  i.e. use the PRINT statements directed along channel N or, and preferably if, you plan to read back data, use the WRITE #N statement instead, e.g.

  20 WRITE #N, X,

- Finally always remember to close your file when all entries are completed. This is done using the CLOSE #N statement, e.g.

1000 CLOSE #N

This statement will close the file, release the file channel N and free all buffer space that the file used.

To access and INPUT data (read from) a sequential file:

- First the file must be OPENED using the OPEN statement:

  10 OPEN "I", #N "Filename"

  the "I" indicates Input from the file; N is the channel number, 1 to 15, you select; "Filename", the name of the file you wish to select as stored on the disk.

- Data stored on the file can be read in sequentially using the INPUT #N, statement, e.g.

  20 INPUT , x,y,z, x$

  The first item stored will be assigned to variable x (assumed to be a real number), the second to y, the third to z, the fourth to x$ (assumed to be a string variable) and so on.

- Finally when you have completed your reading in task, CLOSE the file using the CLOSE statement, e.g.

  1000 CLOSE

Random files

The statement used to OPEN a random file for either input or output is

10 OPEN "R", #N, "filename"

where R refers to random, i.e. to indicate that file is a random one, N refers to the file channel number and "filename" refers to the name of the file.

Examples: using files

- This basic program demonstrates how to open a new sequential file and to output or write data to the file, see lines 30 to 60. The second half of the program accesses the file just created. Line 80 opens the file ready for reading from; the FOR loop, lines 90 to 120, acts to input the data from file to the computer and also display the data read in.

```
10      rem *** Writing to and Reading ***
20      rem   ** from a disk              **
30      open "O",#1,"data1.dat"
```

```
40      for n=1 to 5
50      print £#,n,n*n
60      next n
70      close #1
80      open "I",#2,"data1.dat"
90      for n=1 to 10
100     input #2,x,x2
110     print x,x2
120     next n
130     close #2
```

- The second example given below shows how to create the file "results 1.dat" and input values from the keyboard which are then written and stored in this file. The second half reads back the stored information, processing it as it does to calculate the average. Both the file data and average value are displayed.

```
10      rem *** Sequential file demo ***
20      clearw 2:fullw 2
30      open "O",#1,"results1.dat"
40      for n=1 to 10
50      input "Enter recorded result";x
60      write #1,x
70      next n
80      close #1
90      print:print
100     rem *** Reading from file results1 ***
110     open "I",#2,"results1.dat"
120     print:print
130     print "Results from FILE"
140     print "----------------"
150     for n=1 to 10
160     input #2,value
170     print value
180     sum =sum+value
190     next n
200     close #2
210     print "average of results =";sum/10
```

- All files are terminated by an EOF (End of File) so, when in reading data from a file, that is reached it can be recognised by the computer and used to terminate the reading. The EOF function is very useful in this respect, especially, as is usually the case, when we do not know the exact number of items in the file. Thus rather than using a FOR loop for inputting data

from a file we can use the WHILE loop with the condition "WHILE NOT EOF(N)" to control the input being fed in; N is the file channel number being used.

For example, the FOR loop in the second example above may be replaced by:

```
150    while not EOF(2)
160    input #2,value
170    print value
180    sum =sum+value
190    wend
```

- Here is an example of keeping a telephone directory on file. The first part of the program allows you to create the file and enter names and numbers. Last "terminating" name is ZZZ. The second half allows you to display the list.

```
10     rem *** Phone no. file ***
20     clearw 2:fullw 2
30     open "O",#1,"phonenos.dat"
40     input "Enter name and no.";x$
50     print "after last entry, type ZZZ"
60     write #1,x$
70     while x$<>"ZZZ"
80     INPUT "Enter name and no.";x$
90     write #1,x$
100    wend
110    close #1
120    print:print
130    rem *** Reading from file phonenos ***
140    open "I",#2,"phonenos.dat"
150    print:print
160    print "Names and numbers on FILE"
170    print "-----------------"
180    input #2,x$
190    print x$
200    while x$<>"ZZZ"
210    input #2,x$
220    print x$
230    wend
240    print "ZZZ is used as terminator"
250    close #2
```

EXERCISE PROBLEMS 6.1

1 Write a program that allows you to enter 10 named items and their prices from the keyboard and then display a list of the complete set of items and prices.

2 Modify the above program so that your output display contains ITEM, PRICE, NUMBER of items, Sub-total of value, i.e. number × price, and at the end, the complete value of all items.

3 Using a two-dimensional array, write a program that will display a complete table of results for 32 students each having sat the same four examinations.

4 Design a program that can accept up to 50 numbers in the range 0 to 100 and sort them in numerical order and display the following:
(a) the maximum, minimum, and middle (mode) value
(b) the data values in the following bands:

band 1: 0 to 25,	band 2: 26 to 50,
band 3: 51 to 75	band 4: 76 to 100

5 Write a program that accepts names and phone numbers from the keyboard and then prints out the complete name/phone number list.

6 Modify the last program so that your printout of names is in alphabetical order.

CHAPTER 7

SOME PRACTICAL PROGRAMS

Introduction and summary

This final chapter presents a number of programs for general use. They are all simply constructed using the BASIC statements of previous chapters and easy to understand. All the programs can be run directly to perform their tasks but can also be easily modified to provide any extra facility you might need.

The programs included cover the following applications: conversion of weights, lengths, currency, etc.; production of standard letters, multiple copies; checking bills; calculation of tax; interest and return on investment; loan repayment calculations.

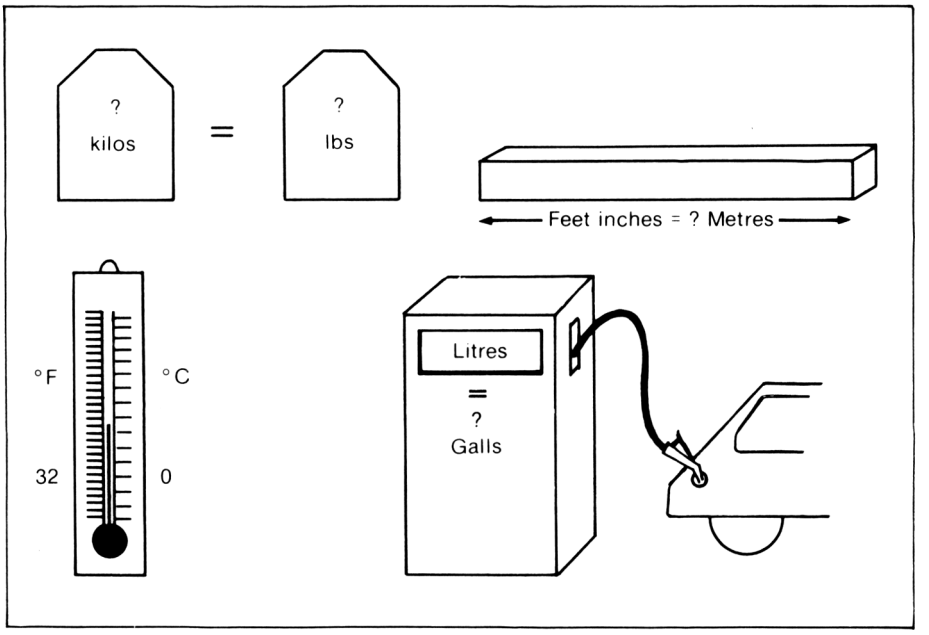


Figure 7.1 *Conversion of units program*

Converting common measures

Figure 7.1 shows some of the common measures that we often need to convert between. The table of Figure 7.2 summarises the conversion factors for converting some commonly used quantities from one system of units, e.g. Imperial, to another, e.g. metric. Rather than remember those, why not use the following program which effects conversions automatically?

```
10    rem *** Conversion of units program ***
20    clearw 2
30    print "to convert;"
40    print " 1 gallons to litres.....enter 1"
50    print " 2 litres to gallons.....enter 2"
60    print " 3 pounds to kilograms.....enter 3"
70    print " 4 kilograms to pounds.....enter 4"
80    print " 5 feet and inches to metres...enter 5"
90    print " 6 metres to feet and inches...enter 6"
100   print " 7 Degrees C to F.............enter 7"
110   print " 8 Degrees F to C.............enter 8"
120   print:input "Now enter your choice";n
130   print:print
140   on n goto 160,180,200,220,240,280,320,340
```

To convert units X to Y multiply by	X	Y	To convert units Y to X multiply by
4.0469×10^{-3}	acres	square kilometres	247.1
2.8317×10^{-2}	cubic feet	cubic metres	35.315
0.3048	feet	metres	3.2808
4.546	gallons	litres	0.21998
0.7457	horse power	kilowatts	1.3410
25.4	inches	millimetres	0.03937
1.6093	miles	kilometres	0.62137
0.35402	miles/gallon	kilometres/litre	2.8247
0.56825	pints	litres	1.7598
0.45359	pounds	kilogrammes	2.2046
0.83613	square yards	square metres	1.1960
1016	tons	kilogrammes	9.8421×10^{-4}

Figure 7.2 *Conversion table for some common quantities*

```
150   print "Incorrect entry":goto 120
160   input "Enter no. of galls to be converted";x
170   print x;" galls =";x*4.546;" litres":end
180   input "Enter no. of litres to be converted";x
190   print x;" litres";x*0.21998;" galls":end
200   input "enter no. of pounds";x
210   print x;" pounds = ";x*0.4536;" kilos":end
220   input "Enter no. of kilos ";x
230   print x;" kilos =";x*2.2046;" pounds":end
240   input "Enter no. of feet";f
250   input "Enter no. of inches";i
260   print f;" feet  ";i;" inches =";
270   print (f+i/12)*0.3048;" metres":end
280   input "Enter no. of metres =";x
290   print x;" metres =";x*3.2808;" feet";
300   print " = ";int(x*3.2808);" feet";
310   print (x*3.2808-int(x*3.2808))*12;" in.":end
320   input "Enter no. of degrees C";x
330   print x;" degrees C =";x*9/5+32;"
        degrees F":end
340   input "Enter no of deg. F";x
350   print x;" deg. F =",(x-32)*5/9;" deg. C":end
360   print "Another conversion ? Yes or no ";
```

On running the program you will first be asked to input the conversion required – the program caters for eight cases, more, of course, may be added. After this all you have to do is input the quantity and the conversion is carried out immediately.

Currency conversions

Here are two useful programs for converting pounds sterling to other currencies and vice-versa. The first program provides a direct means of carrying out the conversion but does require you to enter the exchange rate first:

```
10      rem *** Currency Conversion ***
20      rem short program version    **
30      clearw 2:fullw 2
40      print "For  to other currency.....enter 1"
50      print "For Other currency to .....enter 2"
60      input "Enter 1 or 2 now";n%
70      input "Enter current exchange rate,  =";
          rate
80      print:print
90      if n%=1then  120
100     if n%=2 then 150
110     if n%>2 then print "Incorrect entry":
          goto 40
```

Figure 7.3 *Currency exchange calculation programs*

```
120    input "Enter number of s";p
130    print " ";p; " =";p*rate
140    end
150    input "Amount of other currency";a
160    print a;" currency =";a/rate
170    end
```

The second program contains all the required data within the program for 20 European and American countries. On running the program the 20 countries and corresponding exchange rates are displayed. Update this information if required by amending the DATA statements.

To carry out your conversion all you have to do is to input the country's name (remember, use capital letters), select 1 or 2 for type of conversion and then input the amount. The result is displayed immediately.

```
10     rem *** Currency Conversion ***
20     rem * Display of data stored *
30     clearw 2:fullw 2
40     dim land$(20):dim rate(20)
50     print "Country";tab(20)"Exchange rate"
60     print "------------------------------"
70     for n=1 to 20
80     read land$(n),rate(n)
90     print land$(n),rate(n)
100    next n
110    rem ** Conversion calculations **
120    input "Enter country for exchange";country$
130    for n= 1 to 20
140    if land$(n)=country$ then select.no=n
150    next n
155    print
160    print "For  to other currency...enter 1"
170    print "For other way round.......enter 2"
180    input "Enter your choice";k%
185    print
190    input "enter amount of money for
         conversion";a
200    if k%=1 then 220
210    if k%=2 then 240
220    print " ";a;" = ";a*rate(select.no)
230    end
240    print a;" currency of ";land$(select.no);
250    print " =  ";a/rate(select.no)
```

```
260    data AUSTRALIA,2.065,AUSTRIA,23.65
270    data BELGIUM,68.72,CANADA,2.103
280    data DENMARK,12.44,FRANCE,10.71
290    data GERMANY,3.36,HONG KONG,11.82
300    data IRELAND,1.103,ITALY,2302
310    data JAPAN,258,HOLLAND,3.79
320    data NORWAY,10.73,PORTUGAL,216.8
330    data SPAIN,213.4,SWEDEN,10.81
340    data SWITZERLAND,2.81,USA,1.527
350    data GREECE,202,YUGOSLAVIA,495
```

Multiple copies and standard letters

It is a fairly easy matter to construct programs to print out multiple copies of memos, standard letters, etc. Obviously you require access to a printer to direct your program output to be "printed" as hard copy. The Atari has a standard parallel port interface and it is a simple matter to connect up a printer.

Here are two useful programs. The first provides a means of obtaining the desired number of copies by inputting the required number. The program is set up for display in the output window. To direct output to the printer change all print commands to LPRINT.

```
10     REM *** Production of number of copies ***
20     clearw 2
30     input "enter number of copies required";n
```

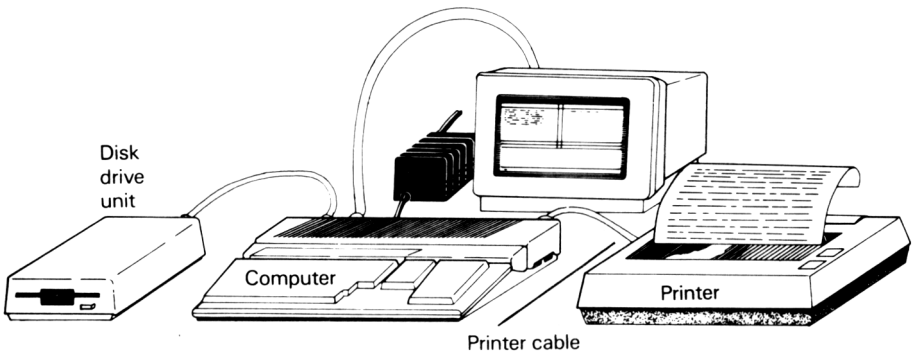


Figure 7.4 *Programs for producing multiple copies, standard letters, etc.*

```
40      print "Dispay follows"
50      print "Remember, use LPRINT for printer
          output"
60      for t=1 to n
70      print "Copy ";t
80      print "******* BORI BAZAAR *******"
90      print "--------------------------"
100     print "....details................"
110     print "..........................."
120     print "..........................."
130     print "........end................"
140     print:print
150     next t
```

The second program illustrates how a standard letter can be produced with the ability to insert common information such as dates, times, places, name, etc. A further advantage is that names to whom the letter is to be send can be entered from the keyboard, stored in an array and inserted automatically as the letter is being printed.

The program has been written to output all PRINT data to the screen so you can first check the letter contents and ensure all input data has been correctly inserted. When the checks have been made and the program is operating correctly change all the relevant "letter" PRINT statements to the printer output form, i.e. use LPRINT.

```
10      rem *** Standard letter ***
20      clearw 2
30      print "Enter the following for inclusion"
40      print "in standard letter"
50      print
60      input "DATE of sending letter ";date$
70      input "DATE of meeting ";meet.date$
80      input "TIME of meeting ";time$
90      input "ROOM to be used ";room$
100     rem * Inputting names for distribution *
110     dim n$(10):rem array for storing names
120     for m=1 to 10
130     print "Enter name ";m;:input n$(m)
140     next m
150     rem ***The standard letter ***
160     for t=1 to 10
170     print:print
180     print tab(20);" RGM Satellite Link
          Computers"
```

```
190    print tab(20);"  Leyden, Sussex  NWFP"
200    print:print
210    print tab(25) date$
220    print tab(4);"Dear ";n$(t);","
230    print "The next seminar on the topic of"
240    print "software for softies is on ";
         meet.date$; "."
250    print "The time of our meeting is ";
         time$;"."
260    print "Tea will be served in ";room$;"."
270    print
280    print tab(25);"PIERS JAMES"
290    print TAB(20);"SENIOR INSTRUCTOR"
300    print
310    print "-----------------------------"
320    next t
```

Checking bills

Here is the framework of a program which allows you to check your domestic bills. You are asked to input the relevant data, such as present and previous meter readings, rate in the pound levied, etc., and you can store constant data such as standing charge, unit cost, VAT, etc., in DATA statements.

Three examples are included: house rates, electricity bills and gas bills.

```
10     rem *** Bill checking program ***
20     clearw 2:fullw 2
30     print "For house rates.......enter 1"
40     print "For electricity.......enter 2"
50     print "For gas...............enter 3"
60     input "Enter section now";n
70     if n>3 then ? "Incorrect entry":goto 30
80     clearw 2
90     on n goto 100,190,360
100    rem **** House rates ****
110    print "House rates check"
120    print "-----------------"
130    rate.val=467:rem rateable value
140    input "Enter current rate pence in the  ";r
150    print
160    print "Annual rates = ";rate.val*r/100
170    print "----------------------------"
```

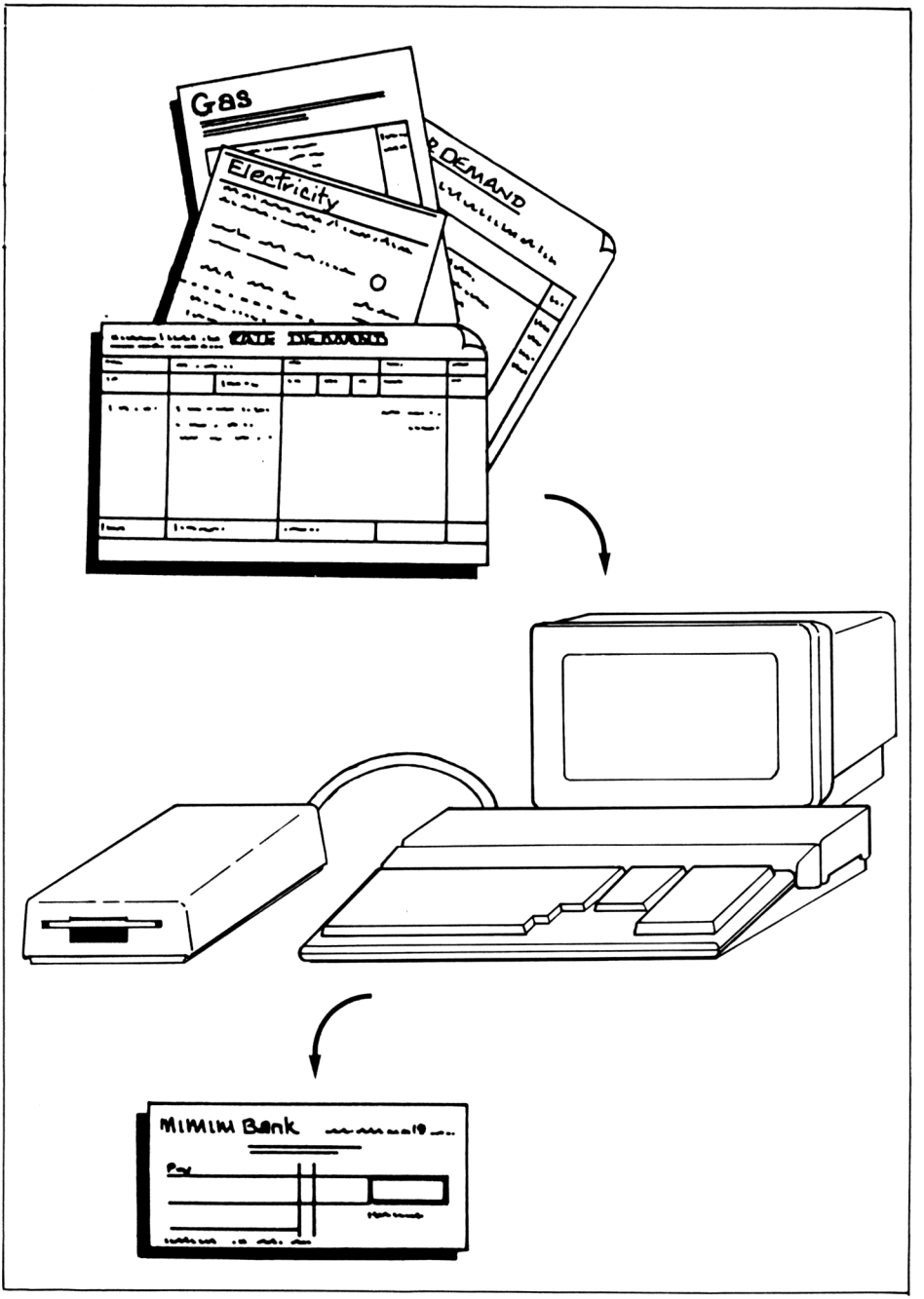


Figure 7.5 *A program for checking domestic bills*

```
180   end
190   rem *** Electricity bill ***
200   clearw 2
210   print "----------------------"
220   print "Electricity bill check"
230   input "Current electricity meter
        reading";ccc
240   input "Previous meter reading";ppp
250   unit.cost=4.96:rem current unit cost in
        pence
260   st.charge=7.45:rem current standing charge
270   vat=0:rem zero vat
280   print (ccc-ppp);" units at ";unit.cost;
        "p = ";
290   print (ccc-ppp)*unit.cost/100
300   print "Standing charge = ";st.charge
310   total=st.charge+(ccc-ppp)*unit.cost/100
320   print "vat at ";vat; " % = ";vat*total/100
330   print "-----------------------------------"
340   print "Totalpayable = ";total*(1+vat/100)
350   end
360   rem *** Gas bill check ***
370   clearw 2
380   print "Gas bill check"
390   print "--------------"
400   therm.f=1.032:rem current thermo-factor
410   print "Enter current and previous meter"
420   input "readings ";ccc,ppp
430   unit.c=35.2:stand.c=9.9:vat=0
440   print (ccc-ppp);" units at ";unit.c;
        "p = ";
450   print (ccc-ppp)*unit.c/100
460   total =(ccc-ppp)*unit.c/100+stand.c
470   print "Standing charge = ";stand.c
480   print "vat at ";vat; "% = ";total*vat/100
490   print "-----------------------------------"
500   print " Total payable = ";total*(1+vat/100)
510   end
```

Income tax calculation

This program enables income tax to be easily calculated. It contains the rates for the 1986-87 tax year so these will require updating when the tax rates are changed. All you have to do is to enter gross income and total allowances, the program then displays the total tax due.

```
10      rem *** Income tax program ***
20      clearw 2:fullw 2
30      print "Tax rates 1986-87"
40      print "-----------------"
50      print "1-17200",tab(20);"29 %"
60      print "17201-20200",tab(20);"40 %"
70      print "20201-25400",tab(20);"45 %"
80      print "25401-33300",tab(20);"50 %"
90      print "33301-41200",tab(20);"55 %"
100     print "Over 41200",tab(20);"60 %"
110     rem tax band constants
120     b1=17200:b2=20200:b3=25400
130     b4=33300:b5=41200
140     print
```

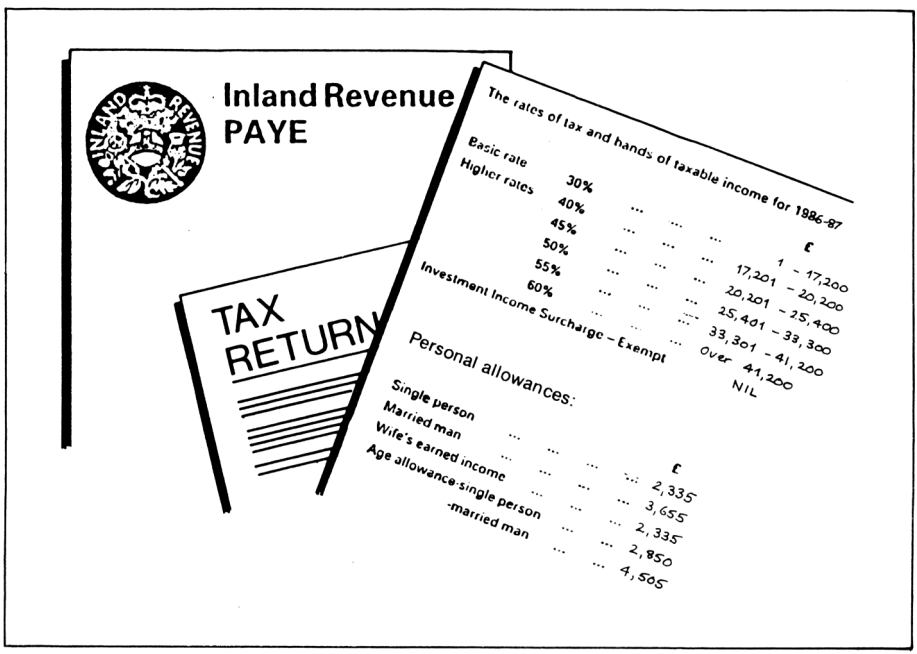


Figure 7.6 *A program for income tax calculations*

```
150   input "Enter gross income";income
160   input "and total allowances";allow
170   taxinc=income-allow
180   if taxinc>0 and taxinc<=b1 then 240
190   if taxinc>b1 and taxinc<=b2 then 260
200   if taxinc>b2 and taxinc<=b3 then 280
210   if taxinc>b3 and taxinc<=b4 then 300
220   if taxinc>b4 and taxinc<=b5 then 330
230   if taxinc>b5 then 360
240   tax=taxinc*0.29
250   print:print "Tax due = ";tax :end
260   tax=b1*0.29+(taxinc-b1)*0.4
270   print:print "Tax due = ";tax:end
280   tax=b1*0.29+(b2-b1)*0.4+(taxinc-b2)*0.45
290   print:print "Tax due = ";tax:end
300   tax =b1*0.29+(b2-b1)*0.4+(b3-b2)*0.45
310   tax=tax+(taxinc-b3)*0.5
320   print:print "Tax due = ";tax:end
330   tax=4988+1200+2340+3950
340   tax=tax+(taxinc-b4)*0.55
350   print:print "Tax due + ";tax:end
360   tax=16823+(taxinc-b5)*0.6
370   print:print "Tax due = ";tax:end
```

Interest and return on investment

Two useful programs for calculations of interest and value of investment are given below. The first can be used to calculate how your money grows in a regular monthly savings plan provided by most Building Societies, Banks and Insurance groups. You input whatever monthly sum you wish to save monthly, the current interest rate and the time in years for your savings plan. The program displays the accrued value of your savings at six-month intervals. Interest is assumed to be compounded every six months as is usually the case for most savings plans. For this reason use is made of the MOD operator (see line 110). MOD provides the remainder after division of two numbers. Thus n MOD 6 = 0 when n = 6, 12, 18, etc., and hence line 110 is only actioned when n is a multiple of 6.

```
10    rem *** Interest on regular savings ***
20    clearw 2
30    input "Enter amount to be saved each
        month";pm
40    input "Rate of interest % given";r
50    input "Enter period of investment, years";t
```

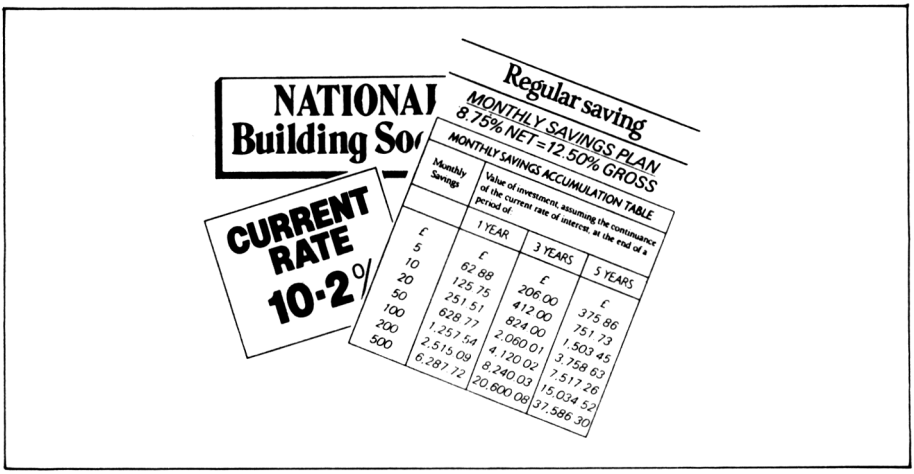

NATIONAL
Building Soc
CURRENT RATE 10·2%
Regular saving
MONTHLY SAVINGS PLAN
8.75% NET = 12.50% GROSS
MONTHLY SAVINGS ACCUMULATION TABLE
Monthly Savings
Value of investment, assuming the continuance of the current rate of interest, at the end of a period of
1 YEAR
3 YEARS
5 YEARS
£ 5 10 20 50 100 200 500
£ 62.88 125.75 251.51 628.77 1,257.54 2,515.09 6,287.72
£ 206.00 412.00 824.00 2,060.01 4,120.02 8,240.03 20,600.08
£ 375.86 751.73 1,503.45 3,758.63 7,517.26 15,034.52 37,586.30

Figure 7.7 *Programs to calculate returns on regular savings and lump-sum investment*

```
60      print "month","accrued value"
70      print "-----------------------------"
80      for n= 1to 12*t
90      p=p+pm
100     i=i+p*r/1200
110     if n mod 6 = 0 then p=p+i:i=0:?n,p
120     next n
130     print "-----------------------------"
140     print;print
150     print "Total value of investment"
160     print "after "t;" years =";p
```

This second program provides a table of values showing how a lump sum investment grows. All you have to do on running the program is to input the lump sum invested, rate of interest, number of years of investment and whether interest is compounded every 6 or 12 months.

```
10      rem ** Interest on lump sum investment **
20      clearw 2
30      input "Enter lump sum invested";p
40      input "Rate of interest % given";r
50      input "Enter period of investment, years";t
60      print
70      print "if interest is compounded every 6
          months"
```

```
80      print "enter 6, if only once a year,
          enter 12"
90      input "Enter 6 or 12 now";g
95      print:print
100     print "month";tab(8) "accrued value";
110     print tab(23)"interest for period"
120     print "-----------------------------------"
130     for n= 1 to 12*t
140     a=p*(1+r/1200)
150     i=i+(a-p)
160     if n mod g = 0 then 170 else 200
170     p=p+i
180     print n;tab(8)p;tab(23)i
190     i=0
200     next n
210     print"-----------------------------------"
```

Loan repayment

The program given below allows you to work out immediately a statement of how your loan repayments are progressing. Repayments are assumed to be made on a regular monthly basis.

On running the program you are first asked to input the amount of the loan, the monthly repayment you wish to make and the interest rate you are going to be charged. The program then provides a display of a table of total repayments made and outstanding debt on a month-by-month basis until the loan is paid off.

```
10      rem *** Loan repayment program ***
20      clearw 2
30      input "Enter amount of loan";loan
40      input "Rate of interest % p.a charged";r
50      input "Enter proposed monthly repayments";
          repay
60      if loan*r/1200 >repay then 70 else 90
70      print "Repayment is insufficient to cover
          interest"
80      print "Increase your repayment":goto 50
90      month=0:print:print
100     print "month";tab(8) "total repaid";
110     print tab(23)"outstanding debt"
120     while loan>0
130     month=month+1
```

Figure 7.8 *A program for loan repayment calculations*

```
140    loan=loan*(1+r/1200)-repay
150    print month,repay*month,loan
160    wend
170    p=p+i
180    print "-----------------------------------"
190    print "Loan is paid off in ";month;
         " months"
200    print "total repayments =";
         repay*month+loan
```

APPENDIX 1

ANSWERS TO EXERCISE PROBLEMS

EXERCISE PROBLEMS 1.1

```
(1) 1332
(2) 160800
(3) 9.99514349
(4) 862.95
(5) 49.17647
```

EXERCISE PROBLEMS 1.2

```
(1)
(a)
PRINT  56.7/3.3  [ENTER] or  [RETURN]
17.1818                        ...answer displayed
(b)
PRINT "56.7/3.3=;56.7/3.3 [ENTER] or [RETURN]
(c)
PRINT  USING  "##.##";  56.7/3.3
[ENTER] or  [RETURN]
17.18                          ...result displayed
```

```
(2)
PRINT 4.62*3.8,57.33/6.3,42378.6-33499.
[ENTER] or  [RETURN]
4.62   9.1   8878.9          ...results displayed
```

EXERCISE PROBLEMS 1.3

```
(1) 4.72944
(2) 726000
(3) 6.37690E-04
(4) 8.695652
(5) 6.294135E-03
```

EXERCISE PROBLEMS 2.1

```
(1)
(a)

10    REM *** names and birth dates ***
20    CLEARW 2
30    PRINT "Name","Birth date"
40    PRINT
50    PRINT "John","10.11.67"
60    PRINT "Mary","14. 7.69"
70    PRINT "Peter"," 2. 3.74"

(b)

10    REM *** Items and price list ***
20    CLEARW 2
30    PRINT "item","Price"
40    PRINT
50    PRINT "Butter","0.92 per kilo"
60    PRINT "Cheese","1.14 per lb"
70    PRINT "Bacon","1.25 per kilo"

(c)

10    REM *** League table ***
20    CLEARW 2
```

```
30   FULLW 2
40   PRINT TAB(20)"*** North Division 1 ***"
50   PRINT
60   PRINT TAB(20)"P";TAB(24)"W";TAB(28)"D";
70   PRINT TAB(32)"L";TAB(36)"F";TAB(40)"A";
80   PRINT TAB(44)"POINTS"
90   PRINT
100  PRINT "Woodside Rovers";TAB(20)"41";
       TAB(24);"22";
110  PRINT TAB(28)"13";TAB(32)"6";TAB(36)"82";
120  PRINT TAB(40)"31";TAB(44)"79"
130  PRINT "Old Polyonians";TAB(20)"41";
       TAB(24)"20";
140  PRINT TAB(28)"14";TAB(32)"7";TAB(36)"71";
150  PRINT TAB(40)"39";TAB(44)"74"
```

2
(a)and(b)

```
10   REM *** Volume of cuboid ***
20   CLEARW 2
30   b=2.27
40   l=5.74
50   d=1.86
60   PRINT "volume =";b*l*d
70   PRINT "----------------"
80   REM *** Volume of cylinder ***
90   PRINT:PRINT
100 r=8.63
110  h=12.4
120  PRINT "Volume of cylinder =";3.142*r*r*h
130  PRINT "--------------------------------"
```

(3)

```
10   REM *** Calculations problem 3 ***
0   CLEARW 2
30   PRINT "Answer 1 =";3.4*5.67-6.21*4.98
40   PRINT:PRINT
50   PRINT "Answer 2 =";5.37^2+4.46^2
60   PRINT:PRINT
70   p=525
80   v=72
```

```
90    c=1.4
100   PRINT "Answr 3 =";p*v^c
```

(4)

```
10    REM *** Simple average program ***
20    CLEARW 2
30    sum=20.7+14.5+12.9+13.6
40    PRINT "Average =";sum/4
50    PRINT "-----------------"
```

```
Answer = 15.425
```

(5)

```
10    REM *** Compound interest program ***
20    CLEARW 2
30    p=100:t=5:r=11
40    a=p*(1+r/100)^t
50    PRINT "Interest =";USING "***.***";a-p
60    PRINT "for p=";p,"t=";t,"r=";r
70    PRINT:PRINT
80    p=25:t=32:r=7.5
90    PRINT "Interest =";USING "****.**";a-p
100   PRINT "for p=";p,"t=";t,"r=";r
```

```
Answers=(a)685.06 (b)227.06
```

EXERCISE PROBLEMS 3.1

(1)

```
10    REM *** Personal data ***
20    INPUT "Enter your name";n$
30    INPUT "       your age ";a$
40    INPUT "    and address";add$
50    CLEARW 2
60    PRINT n$
70    PRINT a$
80    PRINT add$
```

(2)

```
10    REM *** Calculating average ***
20    CLEARW 2
30    READ x1,x2,x3,x4,x5,x6,x7,x8,x9,x10
40    READ x11,x12,x13,x14,x15,x16,x17,x18
50    READ x19,x20
60    sum= x1+x2+x3+x4+x5+x6+x7+x8+x9+x10
70    sum=sum+x11+x12+x13+x14+x15+x16+x17
80    sum=sum+x18+x19+x20
90    PRINT "Average =";sum/0
100   DATA 52,14,37,67,73,11,75,89,19,24
110   DATA 61,49,12,33,47,55,16,71,81,92
```

Answer 48.9

(4)

```
10    REM *** Conversion program ***
20    CLEARW 2
30    PRINT"to convert feet and in to metres"
40    INPUT "Enter feet,inches";ft,in
50    m=(12*ft+in)*2.54/100
60    PRINT ft;" feet ";in;" in =";m;" metres"
70    PRINT:PRINT
80    PRINT "To convert lbs and oz to kilos"
90    INPUT "Enter lbs,oz";lb,oz
110   PRINT lb;" lb ";oz;"oz = ";kg;" kg"
```

(5)

```
5     REM *** Drawing circle ***
10    INPUT "Enter Centre of circle, x and y";x,y
20    INPUT "Enter required radius, r=";r
30    CLEARW 2
40    CIRCLE x,y,r
```

(6)

```
10    REM *** Drawing triangle ***
20    CLEARW 2:FULLW 2
```

```
30    INPUT "Enter coordinates of A ";x1,y1
40    INPUT "Enter coordinates of B ";x2,y2
50    INPUT "Enter coordinates of C ";x3,y3
60    CLEARW 2:FULLW 2
70    LINEF x1,y1,x2,y2
80    LINEF x2,y2,x3,y3
90    LINEF x3,y3,x1,y1
```

EXERCISE PROBLEMS 4.1

(1)

```
10    REM *** password ***
20    CLEARW 2
30    INPUT "Enter password";pass$
40    IF pass$= "APK3287103" THEN 60
50    PRINT "Incorrect password";END
60    PRINT "Secret message begins, code 2/56£"
70    PRINT "01000110 10001101 00001010"
80    PRINT "01011101 01010101 %% 47/42/18 end"
```

(2)

```
10    REM *** To print copies of memo ***
20    CLEARW 2
30    INPUT "No. copies required";n
40    FOR x=1 TO n
50    PRINT "To all club members"
60    PRINT "********************"
70    PRINT "The annual General Meeting"
80    PRINT "will take place...."
90    PRINT "**************************"
95    PRINT:PRINT:PRINT
100   NEXT x
```

(3)

```
10    REM *** READ and find average ***
20    CLEARW 2
30    INPUT "no. of terms";no.terms
40    FOR =1 TO no.terms
50    READ x
```

```
60    sum=sum+x
70    NEXT n
80    PRINT:PRINT
90    PRINT "Average =;sum/no.terms
100   DATA 12,65,98,23,90,34,77,41,32,88
110   DATA 65,89,43,23,9,48,76,83,21,34
```

(4)

```
10    REM *** Selection of courses of action ***
20    CLEARW 2
30    PRINT "Enter 1...for action A"
40    PRINT "Enter 2...for action B"
50    PRINT "Ente 3...for action C"
60    PRINT "Enter 4...for action D"
70    PRINT "Enter 5...for action E"
80    INPUT n
90    IF n>5 THEN PRINT "incorrect entry";GOTO 80
100   PRINT:PRINT
110   ON n GOTO 120,130,140,150,160
120   PRINT "Action A....";END
130   PRINT "Action B....';END
140   PRINT "Action C....";END
150   PRINT "Action D....";END
160   PRINT "Action E....";END
```

(5)

```
10    REM *** to determine Max and Min values ***
20    CLEARW 2
30    no.terms=10:GOSUB 80
40    no.terms=20:GOSUB 80
50    INPUT "No. of terms to be read";no.terms
60    GOSUB 80
70    END
80    REM *** Subroutine for Max and Min **
90    xmax=-1E+10:xmin=1E+10
100   FOR n=1 TO no.terms
110   READ x
120   IF x>xmax THEN xmax=x
130   IF x<xmin THEN xmin=x
140   NEXT n
150   PRINT  "For ";no.terms;" terms"
```

```
160   PRINT "Max vaue =";xmax
170   PRINT "Min value =";xmin
180   PRINT:PRINT
190   RESTORE
200   RETURN
210   DATA 23,45,78,45,33,29,56,23,45,66
220   DATA 34,89,34,71,34,41,56,78,98,21
230   DATA 121,86,4,67,98,34,67,92
```

(6)

```
10    REM *** program for question 6 ***
15    CLEARW 2
20    PRINT "Entry of names and addresses;"
30    PRINT "you will be asked to check each one"
40    PRINT:PRINT
50    PRINT "After last entry terminate by "
60    PRINT "typing in ZZZ"
70    x$=" List of names and addresses"
80    WHILE x$<>"ZZZ"
90    GOSUB 120
100   WEND
110   END
120   REM *** Subroutine for entry and check ***
130   INPUT "Enter Name and address";x$
140   PRINT :PRINT
150   PRINT "Please check!"
160   PRINT "If correct enter.....y"
170   PRINT "If incorrect enter...n"
180   INPUT "Enter y or n";ans$
190   IF ans$="y" THEN 210 ELSE 200
200   PRINT "Please enter again":GOTO 130
210   PRINT x$
220   RETURN
```

EXERCISE PROBLEMS 5.1

(2)

```
10    REM *** To generate 16 random nos ***
20    REM *** in range 1 to 55 ***
30    CLEARW 2
```

```
40    FOR n=1 TO 16
50    y=RND(n)
60    PRINT INT(55*y)+1
70    NEXT n
```

(3)

```
10    REM *** User defined functions ***
15    CLEARW 2
20    REM ** function for average of 3**
30    DEF FNAVERAGE(a,b,c)=(a+b+c)/3
40    a=78.54:b=42.74:c=82.64
50    PRINT FNAVERAGE (a,b,c)
60    PRINT FNAVERAGE (88.7,33.3,66.6)
70    PRINT FNAVERAGE (4,6,5)
80    PRINT:PRINT
90    REM ** volume of cylinder **
100   DEF FN vol(r,h)=PI*r*r*h
110   r=7:h=21:PRINT FNvol(r,h)
120   PRINT FNvol(4.5,8.9)
```

(4)

(a) 30
(b) 25
(c) 2

(5)

```
(a) 90 44 59 57
    45 61 42 33
(b) v £ ; V
```

EXERCISE PROBLEMS 6.1

(1)

```
10    REM *** To enter and display list ***
20    LEARW 2
30    DIM name$(10):DIM PRICE(10)
40    FOR N=1 TO 10
50    INPUT "name of item";name$(n)
60    INPUT "price of item";price(n)
70    NEXT n
```

```
80    CLEARW 2
90    PRINT "item";TAB(15)"price"
100   PRINT
110   FOR n=1 TO 10
120   PRINT name$(n);TAB(15)price(n)
130   NEXT n
```

(2)

```
10    REM *** Problem 2: display list....etc ***
20    CLEARW 2
30    DIM name$(10):DIM price (10)
40    DIM no(10)
50    FOR n=1 TO 10
60    INPUT "name of item";name$(n)
70    INPUT "price of item";price(n)
80    INPUT "no.of items";no(n)
90    NEXT n
100   CLEARW 2
110   PRINT "item";TAB(15)"price";
120   PRINT TAB(22)"NO.ITEMS";TAB(30)"VALUE"
130   PRINT
140   FOR N=1 TO 10
150   PRINT NAME$(N);TAB(15)PRICE(n);
160   PRINT TAB(22)no(n);TAB(30)price(n)*no(n)
170   sum=sum+price(n)*no(n)
180   NEXT n
190   PRINT:PRINT
200   PRINT "Total value =";sum
```

(3)

```
10    REM *** Display of exam results ***
20    CLEARW 2
30    class.size=3
40    DIM name$(class.size)
50    DIM marks(32,4)
60    FOR n=1 TO class.size
70    INPUT "name ";name$(n)
80    FOR j=1 TO 4
90    PRINT "mark ";j;:INPUT marks(n,j)
100   NEXT j
110   NEXT n
```

```
120   FOR n=1 TO class.size
130   PRINT name$(n);TAB(12);
140   FOR j=1 TO 4
150   RINT marks(n,j);
160   NEXT j
170   PRINT
180   NEXT n
```

(4)

```
10    REM *** Solution problem 4 ***
20    CLEARW 2
30    INPUT "No. of terms";no.terms
40    DIM x(50)
50    FOR n=1 TO no.terms
60    READ x(n)
70    NEXT n
80    REM *** Sort routine ***
90    FOR k=1 TO no.terms-1
100   FOR l=k+1 TO no.terms
110   IF x(1)>x(k) THEN 150
120   temp=x(l)
130   x(l)=x(k)
140   x(k)=temp
150   NEXT l
160   NEXT k
170   PRINT:PRINT
180   PRINT "Max value =";x(no.terms)
190   PRINT "Min value =";x(1)
200   mid=INT(no.terms/2)
210   PRINT "Mid value =";x(mid)
220   PRINT:PRINT
230   n=1
240   PRINT "Values in 0 - 25 band;"
250   WHILE x(n)<=25
260   PRINT x(n);
270   n=n+1
280   WEND
290   PRINT:PRINT:PRINT "Values in 25 -50 band:"
300   WHILE x(n)>25 AND x(n)<=50
310   PRINT x(n);:n=n+1:WEND
320   PRINT:PRINT:PRINT "Values in 51 -75 band:"
330   WHILE x(n)>50 AND x(n)<=75
```

```
340   PRINT x(n);:n=n+1:WEND
350   PRINT:PRINT:PRINT "Values in 76 -100 band:"
360   WHILE x(n)>75 AND x(n)<=100
370   PRINT x(n);:n=n+1:WEND
380   DATA 78,23,67,34,23,90,2,61,9,12
390   DATA 44,82,38,67,12,31,8,44,72,56
400   DATA 17,74,93,11,67,52,83,91,62,10
410   DATA 97,32,34,63,73,39,93,32,31,13
420   DATA 56,28,94,34,81,76,7,82,9,43
```

(5)

```
10    REM *** Name and phone no. list ***
20    CLEARW 2
30    DIM NAME.NO$(50)
40    INPUT "No. of names";nmax
50    FOR n=1 TO nmax
60    INPUT "Name and phone no.";name.no$(n)
70    NEXT n
80    CLEARW 2:FULLW 2
90    PRINT "Names and phone numbers"
100   PRINT
110   FOR n=1 TO nmax
120   PRINT name.no$(n)
130   NEXT n
```

(6)

Use above program with addition of alphabetic sort routine

APPENDIX 2

SUMMARY OF ST BASIC KEYWORDS

Given below in alphabetical order are brief definitions of the main keywords used in ST BASIC to execute commands, to provide a given function and generally used in the formation of program line statements.

ABS(X) returns the absolute value of X, i.e. the magnitude of X. X can be any number or numerical expression, e.g. ABS(−10.4) gives 10.4 ; ABS(4.2*−2) gives 8.4.

ASC(X$) or **ASC("string")** returns the ASCII code value of the first character of X$ or the string, e.g. ASC("APPLE") gives 65 (note 65 is the ASCII code for A)

```
10 X$ = "12345"
20 PRINT ASC(X$)
```

gives 49 (note 49 is the ASCII code for 1).

ATN(X) gives the arctan or $\tan^{-1}$ of X in radians, e.g. ATN(1.0) gives 0.785398 (Note: 0.785398 = $\pi/4$ rad = 45 degrees).

AUTO generates automatic line number in steps of 10, beginning at 10 each time the RETURN or ENTER key is pressed.

AUTO start line no., increment

sets automatic line number with specific start and increment, e.g. AUTO 100,5 gives first line at 100 and subsequent lines at 105,110,. . ., etc.

BREAK causes the program currently running to stop execution after every line. Program line and any output are displayed. Pressing RETURN will cause the next line to be executed.

BREAK 100. . .program stops execution after completing line 100
BREAK 100-200. . .program execution stops at 100 and over range to 200
UNBRAKE command cancels BREAK command
To exit BREAK and return to command window type in STOP or END.

CHR$(N) returns the ASCII character corresponding to the number N, N lies between 0 and 256; CHR$(N) is the inverse operation to ASC(X$), e.g. CHR$(65) gives A, as 65 is the ASCII code for A.

CINT(N) rounds number or numeric expression N to the nearest whole number, e.g. CINT(−5.6) gives −6 ; CINT(49.23) gives 49.

CIRCLE x,y,r draws a circle centre x units along screen and y units down screen with radius r units; units are in pixels. CIRCLE x,y,r start angle, stop angle draws part of the circle, i.e. an arc, between the specified angles. The latter are specified in degrees × 10, e.g. CIRCLE 300,150,100,0,900 would draw the arc of circle centre (300,150) and radius 100 between 0 degrees (horizontal axis pointing to right) and 90 degrees (directly upwards), i.e. in this case the first quadrant of the circle.

CLEAR "clears" all program data from memory without erasing the currently stored program.

CLEARW N command used to clear the screen windows where N is

0 for the edit window
1 for the list window
2 for the output window
3 for the command window

e.g. CLEARW 2 would clear all images from the output screen window.

CLOSE #N closes a file which has previously been opened with the OPEN command and releases the number and buffer space the file had occupied; N must lie within range 0 to 15.

CLOSEW N closes the edit window when N = 0, the list window when N = 1, the output window when N = 2, the command window when N = 3.

COLOR text colour no., fill colour no., plot colour no., fill style no 1, no. 2 sets colour of text, fill or background, line with last two numbers defining a number of different styles for the fill colour.

CONT is used to CONTinue execution after program has been stopped by a BREAK or a STOP statement.

COS(X) returns the cosine of X where X is expressed in radians, e.g. COS(1.0472) returns 0.5; 1.0472 rad = 1.0472 $\times 180/\pi = 30°$, i.e. cos 30° = 0.5.

DATA is used to store data in programs and is used in conjunction with the READ statement, e.g.

```
10 READ X, Y, Z
20 PRINT X, Y, Z
30 DATA 22.4, 62.7, 99.3
```

In the READ statement 10, X would be allocated the first value in the DATA statement, Y the second and Z the third. So the output at line 20 would be 22.4, 62.7, 99.3.

DEF FNfunctionname (parameters) = expression is used to define user-defined functions, e.g.

```
10 DEF FNcirclearea(R) = 3.14159*R*R
20 R = 7.2
30 PRINT FNcirclearea
```

would output at line 30 the circle area $3.14159 \times 7.2^2 = 162.860$.

```
40 DEF FNVECTORLENGTH (X, Y, Z) = SQR(X*X*+Y*Y+Z*Z)
50 X = 10 : Y = 20 : Z = 30
60 PRINT FNFECTORLENGTH (X, Y, Z)
```

Line 40 defines a function with parameters X,Y,Z
Line 60 outputs the value for X = 10, Y = 20, Z = 30 m, i.e. $\sqrt{(30^2+40^2+50^2)}$ =70.7107

DELETE deletes a line or lines in a program, e.g.

```
DELETE 100 deletes line 100
DELETE 100-200 deletes from lines 100 up to and including 200
DELETE -60 deletes all lines from start up to 60
DELETE 190-deletes all lines from 190 to end of program
```

DIM is used to generate arrays in programs, e.g. 10 DIM X(9) creates an array of 9 + 1 real elements X(0),X(1) to X(10). If the DIM statement is preceded by an OPTION 1 statement then nine elements are created, X(1) to X(9), i.e. lowest element starts at 1 not 0.

```
20 DIM Y (12, 15)
```

creates a two-dimensional array of 12 + 1 × 15 + 1 elements (assuming OPTION 0) and

```
30 DIM Z$(20)
```

creates an array of 20 + 1 string variable elements. The numbers within brackets in the DIM statement are the array dimensions; up to four dimensions are allowed for integers, three for real numbers and strings.

DIR, the DIRectory command, is used to list the files stored on disk, e.g. DIR lists all files to the command window. DIR B: *.BAS would list all .BAS files on disk in drive B;the * symbol is used as a "wildcard".

EDIT (which can be abbreviated to ED) gives entry to the EDIT Mode, e.g.

EDIT or ED invokes EDIT mode
ED 30 invokes EDIT mode to begin at line 30.

ELLIPSE x,y,a,b draws an ellipse centre x pixel units along screen, y units down screen with a horizontal axis half-diameter of a, and a vertical axis half-diameter of b. ELLIPSE x,y,a,b, start angle, stop angle draws part of the ellipse between the specified angles which should be expressed in degrees × 10, e.g. ELLIPSE 100,100,50,80 draws whole ellipses centre 100,100, horizontal radius 50 and vertical radius 80. ELLIPSE 100,100,50,80,450,900 draws part of ellipse between 45° and 90°.

END ends program execution, closes all files and returns to command mode, e.g.

```
100  IF X GOTO 1000
.
.
.
1000 END
```

Thus if X is greater than 0 the program is told to "GOTO" line 1000 where the END statement ends the program and also closes any files that may have been used.

EOF(N) the End of File function returns a "true" value (actually 1) when the end of a file number N, where N is the number used in opening the file. Very useful in reading all the data from a file, e.g.

```
100 OPEN "I" 2, "DATA1"
110 WHILE NOT EOF (2)
120 INPUT #2, X: PRINT X
130 WEND
```

This program reads in all the data stored in the file DATA1 from disk to the computer.

ERA erases a file on a disk, e.g. ERA TEST1.BAS deletes the program stored on disk under the file name of TEST1.BAS and ERA B: PROGC1.BAS deletes the file stored under PROGC1.BAS on disk in drive B.

ERASE erases an array so that its memory space can be reclaimed; also used if the array is to be redimensioned, e.g.

```
5  OPTION 1
10 DIM array XY (20,10)
20 FOR M = 1 TO 20
30 FOR N = 1 TO 10
40 READ array XY (M,N)
50 NEXT M : NEXT N
60 REM* Now to redimension the array*
70 ERASE array XY. . . .erases array 1
80 ERASE array XY (20,15). . .redimensions array 1
```

EXP(X) returns e^x (exponential function, e = 2.7183), e.g. EXP(2.1) gives $e^{2.1}$ = 8.16617; e^{-1} = 0.367894.

FIELD is used in conjunction with file handling to allocate variable space when transferring information between random file disks and random file buffers, e.g.

```
FIELD #2, 24 AS word$
```

In this case 2 is the number allocated in opening the file and 24 is the number of bytes allocated to the string variable word$.

```
FIELD #3, 10 AS X$, 20 AS Y$
```

allocates 10 bytes to X$ and 20 bytes to Y$.

FILL x,y "fills" shapes with colours with x,y defining the starting position coordinates for FILL; (x,y) is selected to lie within the bounded region to be filled.

FIX(X) truncates a real number to its whole number part (strips out the decimal part), e.g. FIX (10.74) gives 10 ; FIX (−9.98) gives −9.

FLOAT converts an integer N to a real number where N must lie between −32768 and 32767, i.e. $\pm 2^{15}$. For example,

```
X=FLOAT(97)
```

gives X the value 97 but in real form representation with more memory space.

FOLLOW is used as a debugging tool and "follows" (keeps back) the values of all program variables specified. FOLLOW prints the variable name, its value and the program line as each line is executed, e.g.

```
10 FOR N =2 TO 4
20 X = N*N−1
30 NEXT N
FOLLOW N,X   [RUN]
```

would produce the following display in the command window:

1. N! = 2 at line 10
2. X! = 3 at line 20
3. N! = 3 at line 30
4. X! = 8 at line 20
5. N! = 4 at line 30
6. X! = 15 at line 20
7. N! = 5 at line 20

The FOLLOW command is cancelled by the UNFOLLOW command

The FOR statement:

For N = start value TO stop value STEP increment value

NEXT creates a repetitive loop which steps from the start value by increments given by STEP (if STEP is omitted the increment is 1) to stop value. It is usually used in conjunction with NEXT, e.g.

```
10 FOR N = 1 TO 3 STEP 0.5
20 PRINT N, N*N
30 next
```

will output N and N × N with N starting at 1 in steps of 0.5 and ending when N = 3.

FRE(1) returns the number of unused bytes in memory. FRE requires a dummy argument (number in brackets); use any value, 1 is used as above.

FULLW N sets the 4 basic windows to full screen size: N = 0 the edit, N = 1 the list, N = 2 the output and N = 3 the command window.

GET #N, record number reads a record from a random disk file into a file buffer; N is the file number used in the OPEN statement, the record number is optional and when omitted the next record after the first GET or PUT is fed into the buffer, e.g.

```
10 OPEN 'R', "data1"
20 GET # 1
```

Line 10 opens an existing random file "data 1" and line 20 will feed the first record from the data1 file stored on disk to the computer's file buffer.

GOSUB line number, GOSUB label name The GOSUB statement passes control to the subroutine at the specific line number or label name, e.g.

```
10 GOSUB 500
20 ---
.
.
```

```
500 REM start of Subroutine
510 ---
600 RETURN
```

Statement 10 jumps program execution to line 500 and after executing the routine, line 600 returns control to line 20, immediately after the GOSUB call, e.g.

```
100 GOSUB MESSAGE
110 . . .
.
.
1000 MESSAGE ; print "Feed in data"
1010
.
.
1100 RETURN
```

At line 100 program control jumps to the label MESSAGE at 1000 and then is returned to line 110 by RETURN at line 1100.

GOTO, GOTO line number, GOTO line label passes program execution directly to specified line number or label, e.g.

```
100 input "Interest Rate", R
110 IF R>25 THEN GO TO GETOUT
.
.
200 GETOUT: PRINT  "Excessive, end calculation"
210 END
```

GOTOXY x,y places output cursor position column x and row y, e.g.

```
100 GOTOXY 5,6
110 PRINT "*"
120 PRINT  "Position of *is at column 5, row 6"
```

HEX$(N) returns a string which is the hexadecimal (base 16) representation of the number N, e.g.

```
10 N = 1047
20 PRINT HEX$(N)
```

would output 417, i.e. $417_{16} = 1047_{10}$.

IF statements, for example

```
IF logical expression THEN statement
IF logical expression THEN statement: ELSE statement
```

are used in making decisions. If the logical expression is true, i.e satisfied,

THEN the statement(s) following THEN are executed. If ELSE is also included the logical expression is again evaluated and if true THEN the statement(s) following THEN are executed; if false the statement(s) following ELSE are executed.

INPUT inputs values to computer via keyboard, e.g.

10 INPUT X

When this statement is executed ? appears in output window awaiting entry of value to be assigned to X; type in numeric value and press RETURN;

20 INPUT "Enter value, please"; X

string within quotes also displayed; enter value for X and press RETURN:

30 INPUT "Values for A,B and C"; A,B,C

used for multiple inputs; separate values entered by,

40 INPUT "your name and address"; A$ string variable input

INPUT# N,X works exactly similar to INPUT but in this case input is read from a file which must first be opened, using OPEN"I",#N (note space between INPUT and # appears optional, i.e. INPUT# and INPUT # behave the same, e.g. INPUT #1, X or INPUT# 1,X statement of this form skips all spaces, tabs, line feeds, carriage feeds and assigns first number read in file to X. INPUT #2, X$ or INPUT# 2, X$ reads from the file containing "character data" and assigns this to X$.

INT(X) returns the whole number (integer) part of the number or numeric expression X, e.g.

10 Y% = INT(−99.76)

would assign Y% as the integer −99.

KILL "filename" is used to delete a file stored on disk, e.g. KILL "DATA-1.DAT" deletes the file DATA1.DAT on disk.

LEFT$(A$,N) returns the first N characters of a string counting from left, e.g.

10 A$ − "Peshawar"
20 PRINT (A$,6)

line 20 would display Peshaw, the first six characters in Peshawar.

LEN(A$) gives the length, i.e. number of characters, in a string, e.g.

10 A$ = "Amsterdam Holland"
20 PRINT LEN (A$)

Line 20 displays 17, the number of letters in Amsterdam Holland.

LET used in assignement statements but can be omitted and it is now normal practice not to include LET, e.g. 10 LET x = 40 and 10 x = 40 are identical – x is assigned the value of 40. Similarly 20 LET x $ = "phone no." and 20 x$ = "phone no." are again identical, the string variable x$ is assigned the value "phone no.". Note: string variable values are enclosed in quotation marks in assignment statements.

LINE INPUT is similar to input but allows entry of an entire line of up to 255 characters, e.g.

```
10 LINE INPUT "Enter full name and address"; AD$
20 PRINT AD$
```

On executing line 10 the prompt displayed: "Enter full name and address"; type in up to 255 characters maximum. This string is assigned to AD$ which is displayed by line 20.

LINE INPUT# N, string variable is used to input up to 254 characters from a sequential file disk file and assigns the input to the specified string variable; N is the channel number used in opening the file; all characters are read until a carriage return is reached.

LINEF x_1,y_1,x_2,y_2 draws a line between points (x_1,y_1) and (x_2, y_2) in the output window, e.g. LINEF 0,0,100,150 draws a line from (0,0) to (100,150) in the output window.

LIST is used to display whole or part of the current program listing in the list window, e.g.

```
LIST displays entire program
LIST -100 displays list from start to line 100
LIST 100- displays list from 100 to end of program
LIST 100-200 displays list range between 100 and 200
LIST 50 display single specified line, e.g. 50
```

Note: pressing CONTROL and G keys stops LIST and returns to the command mode.

LLIST is exactly similar to LIST, but the output of listing is directed to printer; the command WIDTH LPRINT N sets line width at printer to be N characters, N up to 72.

LOAD filename command used to LOAD program files from disk into the computer's memory, e.g. LOAD PROG1.BAS would load the file PROG-1.BAS stored on disk to the computer; program can now be run.

LOG(x) returns the natural logarithm of a number (log to base e where e = 2.71828), e.g. LOG (4.6) gives 1.526056.

LOG10(X) returns the logarithm of a number to base 10, e.g. LOG10(1000) gives 3.0000.

LPRINT is exactly similarly to PRINT but directs output to the printer, e.g.

```
10 A$ = "P K PHILIPS"
20 B$ = "14 The Hines, Belevadere"
30 LPRINT A$
40 LPRINT B$
50 LPRINT "--------"
```

Lines 30, 40 and 50 would be output to the printer and recorded as hard copy.

MERGE filename is used to combine or "merge" a program stored on disk with the program currently stored in memory, e.g. MERGE PROC1.BAS would read in the program PROC1.BAS from disk and merge it with the program currently stored in memory.

MID$ (A$, start char.no., length) returns part of a string starting at the specified character number from left and of the length specified, e.g.

```
10 A$ = "My name is Roger Dean"
20 PRINT MID$(A$, 12, 5)
```

Line 20 would output Roger, i.e. the 12th to 12 + 5 = 17th characters.

NAME "old name" AS "new name" is used to rename a file already stored on disk, e.g. NAME "TEST1.BAS" AS "PROG1.BAS" changes the name of TEST1.BAS on disk to PROG1.BAS.

NEW is used to completely clear the current program from the computer's memory, e.g. NEW clears the current program and NEW program name clears the current program but will allow this program to be saved.

NEXT is used as terminator in FOR loops (see FOR statement).

**OCT$** returns a string which is the octal (base 8) equivalent of a decimal number, the latter being rounded up to a whole number before conversion; conversion range −32768 to 32767, e.g. OCT$ (254) gives 376, i.e. $254_{10} = 376_8$.

OLD filename is similar to LOAD, the OLD command loads an existing file from disk.

ON n GOTO and ON n GOSUB, for example,

```
ON n GOTO line no.1, line no.2, line no.3
ON n GOTO label 1, label 2, label 3
ON n GOSUB line no.1, line no.2, line no.3
ON n GOSUB label 1, label 2, label3
```

transfer control to a given line or label depending on the value of number n; if n = 1 program execution jumps to line no.1/label 1, if n = 2 execution jumps to line no.2/label 2 and so on, For example,

```
10  INPUT "Selection value n=1, 2 or 3"; n
20  ON n GOTO 100, 200, 300
.
100 Rem * execution jumps to here if n = 1 *
.
200 Rem * execution jumps to here if n = 2 *
.
300 Rem * execution jumps to here if n = 3 *
```

OPEN is used to open a disk file ready for writing data to or reading data from a file on disk. OPEN "O", #N, "file name" is used to open a sequential file ready for writing to; N is the file channel number to be used, between 1 and 15 and which you select; the name between quotes is the name you wish to give your file for new files or, for existing files, the name of the file you wish to access. OPEN "I", #N,"filename" is used to open an existing sequential file on disk and read in or input its contents to the computer. OPEN "R", #N, "filename" is used for both inputting from and outputting to random access files.

OPENW N is used to open one of the four ST BASIC windows : when N = 0 OPENW O would open the Edit window; N = 1 corresponds to List ; N = 2 to Output and N = 3 to the Command window.

OPTION BASE N is used to set the minimum value of the array element subscript at either 0 or 1, e.g.

```
10 OPTION BASE 1
20 DIM X (100)
```

would define an array of 100 elements, x(1) to x(100),

```
30 OPTION BASE 0
40 DIM X$(20)
```

would define an array of 20 + 1 string variable elements x$(0) to x$(21).

PCIRCLE x,y,r draws a solid (filled-in) circle centre (x,y) and radius r. PCIRCLE x,y,r start angle, stop angle draws a solid segment of circle starting at start angle and ending at stop angle ; these angles are specified in degrees × 10, e.g. PCIRCLE 100,100,80,900,1200 would draw a solid segment centre (100,100), radius 80 starting at 90° and ending at 120°.

PEEK memory location number returns the contents of the specified memory location.

PELLIPSE x,y,a,b draws solid (filled-in) ellipse, centre (x,y), half-diameter a in x-direction, half-diameter b in y-direction. Solid segments can be drawn using ELLIPSE x,y,a,b, start angle, stop angle where start and stop angles are expressed in degrees × 10.

POKE location number, poke data stores a value of data to "poke" into specified memory location.

PRINT outputs data for display in the Output window. Note: ? may be used as an abbreviation for print, e.g. PRINT X and ? X are identical and display the value of X in the Output window. PRINT X$ and ? X$ display the value of string variable X$ in the Output window. PRINT X,Y,Z displays values of X, Y and Z "separated" in 14 space zones. PRINT X; Y; Z displays X, Y, Z with one space separations.

PRINT# N,X is used to output data to disk file via channel N where N is the file channel number declared in the OPEN statement, e.g.

```
10 OPEN "0", #3, "Results.DAT"
20 FOR n = 1 to 100
30 INPUT "Enter result"; x
40 PRINT #3, x
50 NEXT n
```

The FOR loop is used to input 100 results from the keyboard, see line 30, and then output each to the disk file Results.DAT using line 40. (Note: it appears a space can be left bewtween PRINT and the # symbol.)

PRINT USING "###.###" is used for formatting numerical data ouput, e.g. PRINTUSING "##.##"; 4.369*2.7823 displays the result ,12.16, to two decimal places. PRINT USING "##.###^^^^": 7596.45*5729.24 outputs the result in standard form to three decimal places, 4.352E+08, i.e. 4.352×10^8.

PUT #N, record number writes a record from a buffer to a random disk file ; N is the file number used in the OPEN statement. The record number is optional but when included must begin at 1 and proceed in ascending order.

QUIT is used to "quit" ST BASIC and return to the operating system.

RANDOMIZE 0 is used to "seed" a random number generator; used in conjunction with the RND random number function, e.g.

```
10 RANDOMIZE 0
20 FOR N = 1 TO 100
30 PRINT RND;
40 NEXT N
```

will output 100 random numbers between 0 and 0.999999.

READ used to assign data values contained in DATA statements to variables, e.g.

```
10 READ A,B,C
20 PRINT A,B,C
100 DATA 44.6, 52.9, 14.7
```

would assign A = 44.6, B = 52.9, C = 14.7.

REM statements allow inclusion of REMarks in programs. REM statements are skipped over in program execution, e.g.

```
10 REM Program to list results and to
20 REM calculate average and median values
```

RENUM used alone renumbers the current line numbers starting at 10 and in increments of 10.

RENUM new first line, starting line, increment begins renumbering at "starting line" specified with "new first line" number as specified and specified increment, e.g. RENUM 100, 10, 5 would renumber program from original line 10, assign this 100 and then set increments of subsequent lines at 5, i.e. renumbered program would have lines 100, 105, 110. Note: RENUM requires a non-write-protected disk to be in position since its use requires the creation of a BASIC.WRK file for its operation.

REPLACE filename, line no. list is used to replace an old version of a program or file stored on disk with a new or revised version, e.g. REPLACE sends the revised version onto disk replacing the old version under the original name; REPLACE VERSION 2 saves the sources program under VERSION2 rather than the original name.

RESET places the contents of the output window into the graphics buffer, thus allowing the graphics images to be stored on disk or to be restored to the output window after other graphics operations.

RESTORE is used to set the READ pointer at the first item in the first DATA statement contained in the program; if RESTORE is used with a specified line number then the READ pointer is set at the first item of this number or at the first DATA statement at the specified number, e.g. RESTORE 100 sets pointer at first DATA statement at 100 or following 100.

RETURN is used to terminate a subroutine and transfer program execution to the line immediately following the GOSUB call, e.g.

```
100 GOSUB 1000
110. . .
```

```
.
1000 REM beginning of subroutine
.
1200 RETURN
```

At line 100 execution jumps to subroutine beginning at 1000; after execution of routine on reaching line 1200, the RETURN statement returns execution to line 110 immediately following the GOSUB call.

RIGHT$ (A$,N) returns the N rightmost characters of a string, e.g.

```
10 A$ = "Scientific Instrument Centre"
20 PRINT RIGHT$ (A$,6)
```

will output the last six characters, i.e. Centre.

RND and **RND(N)** are used to generate random numbers in the range 0 to 0.999999; used in conjunction with RANDOMIZE. N is a numeric expression and is optional. If N is used and with N posititive the next number in random sequence is returned. If N = 0 the last number in the sequence is returned. If N is negative the random number generator is reseeded and the first number in the new sequence is returned; if N is not used RND acts the same as with N positive, e.g. to generate a set of random numbers in the range 0 to 999:

```
10 RANDOMIZE 0
20 FOR n = 1 to 100
30 X =1000 * RND
40 PRINT INT(X)
50 NEXT n
```

RUN is used to start current program execution from the beginning; RUN line no. runs program from specified line no., e.g. RUN 1000 starts program execution from line 1000. RUN program name clears the currently stored program, then loads the specified program and runs it.

SAVE filename saves specified filename on disk, e.g. SAVE TESTF1.BAS saves the currently stored program on disk under the name TESTF1.BAS. Use of SAVE TESTF2.BAS, 100-200 saves lines 100 to 200 on disk.

SGN (X) returns the sign of the number or numeric expression X; if X = 0 it returns 0.

SIN(X) returns the sine of X, X being specified in radians. Angles specified in degrees must first be converted to radians, e.g.

```
10 input "Angle x in degrees";x
20 x = x *3.14159/180
30 print SIN(X)
```

SOUND controls the three sound channels. SOUND syntax is as follows:

SOUND voice no., volume no., note no., octave no., duration no.

where voice no. 1 to 3 is the number of sound channels used; volume no. controls loudness: 0 = OFF to 15 = loudest; note no. 1 to 12 and octave no. 1 to 8 determine the pitch of a note; duration no. determines the times in 1/50 second units of the note duration.

SPACE$(N) returns a string of N spaces, N range 0 to 255.

SPC(N) is used in conjunction with PRINT to output spaces in a PRINT statement, e.g.

10 PRINT "1" SPC(5) "2" SPC (4) "3"

outputs

1 2 3

i.e. 1 followed by five spaces, then 2 followed by four spaces and then the 3.

SQR(X) returns the square root of the number OR numeric expression X.

STOP is used to STOP program execution and transfer control to the command window: STOP leaves files opened and enters the BREAK mode. Execution can be continued by pressing RETURN or typing in CONT.

STR$(X) returns a string representing the decimal number X, i.e. converts a numeric variable to a string variable representation.

SWAP x,y is used to interchange or swap variables of same type, e.g.

```
10 x = 10 : y = 20
20 PRINT x; y . . .outputs 10 20
30 SWAP x, y
40 PRINT x; y . . .outputs 20, 10
50 A$ = "MEADOWS"
60 B$ = "R.G."
70 SWAP A$, B$
80 PRINT A$, B$ . . . outputs R.G. MEADOWS
```

SYSTEM is a command that causes execution to exit ST BASIC and return to the GEM command level; same as QUIT command.

TAB(n) is used in conjunction with PRINT to move cursor n spaces from left-hand margin, e.g. PRINT TAB (10)"*" will output * in the 10th column of the output window.

TAN(X) returns the tangent of X, where X is specified in radians.

TRACE is used to "trace" program execution line by line, printing each line before executing it in the command window and then displaying any output in the output window. TRACE is cancelled by the command UNTRACE.

TROFF and **TRON** The TRON command "traces" program execution line by line (as TRACE), but displays only the line number in the command window, but does display any results produced in the output window. TRON is cancelled by the command TROFF.

UNBREAK can be used to cancel the BREAK command either completely or for selected lines, e.g. UNBREAK completely cancels BREAK and UNBREAK 10-100 cancels BREAK over specified range.

UNFOLLOW is used to cancel the FOLLOW command either completely or for selected variables, e.g. UNFOLLOWS X,X$ cancels FOLLOW for specified variables.

UNTRACE is used to cancel the TRACE command either completely or over selected line ranges.

VAL(A$) converts the string of digits A$ to a real number, e.g.

```
10 A$ ="123" : A = VAL(A$)
20 B$ = "321" : B = VAL(B$)
30 PRINT A; B; A+B
```

Line 10 defines a string of digits and the second statement converts the string to a real number. Likewise in line 20. Output from line 30 is therefore 123 321 444.

WHILE...WEND loop The WHILE statement at the beginning of the loop sets conditions that if true allows entry to the other loop statements and if false skips over the statements; WEND is used as the terminating statement of the loop e.g.

```
100 INPUT "Measured length";L
110 WHILE L>95 AND L<105
120 COUNT = COUNT + 1
130 INPUT "Enter next length";L
140 WEND
150 PRINT "Reject length"; COUNT
```

The WHILE-WEND loop is continally executed provided L is greater than 95 and less than 100; if a value of L outside this range is input, then the loop is executed and execution proceeds to the line immediately following WEND.

WIDTH is used to set the line width (number of characters per line) of the output window or printer, e.g. WIDTH 25 sets the number of characters displayed per line to 25.

WRITE acts like PRINT but places commas between successive items and displays string values enclosed by quotation marks, e.g.

```
10 A$ = "K P CHIPS" : A = 99
20 B$ ="P K GUMM": B = 5
30 WRITE A$, A, B$, B
```

The WRITE statement produces the output display:

```
"K P CHIPS",99, "P K GUMMS", 5
```

i.e. with strings enclosed by quotes and items separated by commas.

WRITE# N,X,Y acts like PRINT# to output, i.e. write, data to a sequential file which has been previously opened by the OPEN command; N is the channel number used in opening the file. WRITE# is preferable to PRINT# if it is planned to read back data from the file using INPUT# statements. (Note: as for INPUT#, PRINT# a space can be left between WRITE and #.)For example,

```
10 OPEN"I",# 7, "Fencing1"
20 FOR n = 1 to 10
30 INPUT F$, COST
40 WRITE# 7, F$, COST
50 NEXT n
```

Line 10 opens a sequential file called Fencing 1. Within the FOR loop information is input from the keyboard, see line 30, and then written to the disk file using the WRITE# statement of line.

APPENDIX 3

ST BASIC RESERVED WORDS

The following is a list of reserved words used in ST BASIC and these cannot be used as identifiers for variables. If you use a reserved word for an identifier the error message, "Something is wrong", will appear in the command window as soon as you press RETURN.

ABS
ASC
ATN
AUTO
BLOAD
BREAK
BSAVE
CALL
CHAIN
CHR$
CINT
CIRCLE
CLEAR
CLEARW
CLOSE
CLOSEW
COLOR
COMMON
CONT
COS
CVD
CVI
CVS
DATA
DEF FN
DEF SEG
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DIM
DIR
EDIT
ELLIPSE
END

EOF
ERA
ERASE
ERL
ERR
ERROR
EXP
FIELD
FILL
FIX
FLOAT
FOLLOW
FOR
FRE
FULLW
GEMSYS
GET
GOSUB
GOTO
GOTOXY
HEX$
IF
INP
INPUT
INPUT#
INPUT$
INSTR
INT
KILL
LEFT$
LEN
LET
LINE INPUT
LINE INPUT#
LINEF
LIST
LLIST
LOAD
LOC
LOF
LOG
LOG10
LPOS
LPRINT
LSET
MERGE
MID$
MKD$
MKI$
MKS$
NAME
NEW
PEEK
PELLIPSE
POKE
POS
PRINT
PRINT#
PRINT USING
PUT
QUIT
RANDOMIZE
READ
REM
RENUM
REPLACE
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SGN
SIN
SOUND
SPACE$
SPC
SQR
STEP
STOP
STR$
STRING$
SWAP
SYSTAB
SYSTEM
TAB
TAN
TRACE
TROFF
TRON
UNBREAK

UNFOLLOW
UNTRACE
VAL
VARPTR
VDISYS
WAIT
WAVE
WEND
WHILE
WIDTH
WRITE
WRITE#

INDEX

Entries given in capital letters are ST BASIC commands and/or statements.

ABS(X), 99, 165
AND logic expression, 87
Applications, 98
Arrays, 117
  one-dimensional, 119
  OPTION BASE statement, 120
  pictorial representation, 118
  three-dimensional, 120
  two-dimensional, 120
  use and application, 121
ASC function, 112
ASCII code, 112–113
Assignment statements, 44
  LET, 44
ATN(X) function, 103, 165
AUTO, 43, 165

Boolean operators, 87
  examples of use, 88–90
  order of precedence, 88
Brackets, use of, 37
BREAK, 44, 166
Bubble sort routine, 126
  pictorial representation, 128
  programs, 126, 127

Calculations, 20
  conversion calculator (program), 67
  example displays, 21
  powers, use of ^ key, 36
  random numbers, 100
  square roots, 36, 99
  squares and cubes, 66
  temperature conversion, 67
  use of brackets, 37
  volume, 68
Character sets and codes, 112
CHR$ function, 112, 166
CINT(X), 99, 166
CIRCLE, 28, 166
CLEAR, 166
CLEARW, 23, 44, 166

program line statement, 46
CLOSE, 166
Colon, use of in program statements, 45
COLOR, 31, 166
examples of use, 33–36
Comma, in PRINT command, 24
examples of use, 25
Commands *(for individual commands, see under command name):*
fundamental direct mode, 42
COMMAND window, 19
Comparison operators and symbols, 78
CONT (CONTinue), 65, 71, 167
Conversion, denary to hexadecimal, 114
COS(X), 143, 167
Cylinder in perspective (program), 49

DATA, 167
Data input, 65
from arrays, 121
Debugging programs, 60
Decision-making, 76
DEF, 167
DELETE, 43, 167
Deleting programs, 59
Denary-to-hexadecimal conversion, 114
DIM statement, 117, 167
examples of use, 119
one-dimensional arrays, 119
three-dimensional arrays, 120
two-dimensional arrays, 120
DIR, 58, 168
Directory listing, 58

ED, 168
EDIT, 168
Editing programs, 53
cursor control keys use, 55
functions keys use, 55
EDIT mode, 53
displays, 54, 55
EDIT window, 19, 53
ELLIPSE, 28, 168
END, 168
EOF, 168
ERA, 59, 168
ERASE, 169

Exponential functions, 106
EXP(X) function, 106

FIELD, 169
File:
display after "clicking" on, 17
display with Open option, 18
Files, 117
examples of use, 133
random, 133
sequential, 132
use in programs, 131
FILL, 31, 169
examples of use, 33–36
FIX(X), 99, 169
FLOAT, 169
FOLLOW, 169
FOR, 170
FOR loop, 80
examples of use, 80–85
flowchart, 81
FRE, 170
FULLW, 24, 170
Functions:
exponential, 106
logarithmic, 106
mathematical, 98–103
standard, 98
strings and string-slicing, 109–112
trigonometric, 103–107
user-defined, 107–109

GET, 170
GOSUB, 94, 170
GOTO, 90, 171
GOTOXY, 91, 171
Graphics commands:
CIRCLE, 28
COLOR, 31, 33
displays, 29–32
ELLIPSE, 28
examples of, 29–36
FILL, 33
LINEF, 28
PCIRCLE, 31
PELLIPSE, 31

Hard copy, obtaining, 53
Hexadecimal, conversion to from denary, 114
HEX$, 171

Identifiers, 50
IF statements, 76, 171
  examples of use, 78
  flowchart, 77
IF ... THEN statements, 77
  examples of use, 79
  flowchart, 77
IF ... THEN ... ELSE statements, 77
  examples of use, 79
  flowchart, 77
Initials, display of (program), 47
INPUT, 66, 172
  and arrays, 121
Inputting data, 65
  from arrays, 121
Integer variables, 51
INT(X), 172

Jumping, 76

KILL, 172

Labels, use of, 90
LEFT$ function, 109, 172
LEN function, 111, 172
LET, 44, 173
LINEF, 28, 173
LINE INPUT, 173
LIST, 43, 173
LIST window, 19
LLIST, 53, 173
LOAD, 58,173
Logarithmic functions, 106
Logic expressions, 87
LOG(X), 106, 173
LOG10(X), 106, 174
LPRINT, 53, 174
Lower-case usage, 52

MERGE, 59, 174
Merging programs, 59
MID$ function, 110, 174

NAME, 174
Names and phone numbers (program), 46
NEW, 43, 174
NEXT, 174
NOT logic expression, 87
Numeric, or real, variables, 50

OCT$, 174
OLD, 174
ON ... GOSUB, 174
ON ... GOTO, 93, 174
OPEN, 132, 175
OPENW, 175
Operating system:
  initial display, 17
  loading, 16
Operators:
  Boolean, 87
  comparison, 78
OPTION BASE statement, 120, 175
OR logic expression, 87
Output statements, 45
OUTPUT window, 19

PCIRCLE, 31, 175
PEEK, 175
PELLIPSE, 31, 176
Phone numbers and names (program), 46
POKE, 176
Powers, calculation of, 36
PRINT, 15, 20, 45, 176
  "..." in, 24
  comma in, 24
  further applications, 24
  semi-colon in, 24
  TAB function, 24
  use of ? for, 22
PRINT USING, 24
  examples of use, 25, 26
Programs:
  basic points to consider, 41, 42
  beginning to write and run, 39
  data input, 65
  debugging, 60
  deleting, 59

directory listing, 58
editing, 53
fundamental commands and statements, 42
interacting with, 65
loading, 58
merging, 59
saving, 56
statements, 44
see end of this Index for a complete list of sample programs
Programs, practical:
checking bills, 144
converting common measures, 138
currency conversion, 140
income tax calculation, 147
interest calculation, 148
loan repayment, 150
multiple copies, 142
return on investment, 148
standard letters, 142
Programs, simple:
area of triangle, 48
conversion calculator, 67
cylinder in perspective, 49
displaying initials, 47
inputting more than one value, 67
names and phone numbers, 46
squares and cubes, 66
temperature conversion, 67
PUT, 176

QUIT, 176

Random files, 133
RANDOMIZE, 100, 176
Random numbers, 176
READ, 177
READ-DATA statements, 65, 69
and arrays, 121
program examples, 69–71
with RESTORE, 72
Real or numeric variables, 50
REM (REMark), 45, 177
RENUM, 43, 177
Repetition, 76
FOR loop, 80
WHILE loop, 85
REPLACE, 177
RESET, 177
RESTORE, 72, 177
RETURN, 177
RIGHT$ function, 110, 178
RND, 100, 178
RUN, 42, 178

SAVE, 56, 178
Saving programs, 56
displays, 57
Semi-colon, in PRINT command, 24
examples of use, 25
Sequential files, 132
SGN, 178
Sign of numbers, 99
SIN(X), 143, 178
Sorting, 126
alphabetical order, 129
number order, 126
Sort routine, bubble, 126
SOUND, 179
SPACE, 179
SPC, 179
SQR(X), 98, 179
Square roots, 98
Standard functions, 98
Statements, 44
assignment, 44
colon, use of in, 45
GOTO, 90
INPUT, 66
LET, 44
output, 45
PRINT, 45
READ-DATA, 65
REMark, 45
RESTORE, 72
STOP, 65
ST BASIC:
commands and statements, fundamental, 40
display when fully loaded, 40
getting started, 15
keywords summary, 165

loading, 15
reserved words, 182
use of in programs, 39
ST computer system components, 16
STOP statement, 65, 179
Strings, 109
String slicing functions, 109
LEFT$, 109
LEN, 111
MID$, 110
RIGHT$, 110
STR$, 111
VAL, 111
String variables, 51
inputting, 68
STR$ function, 111, 179
Subroutines, 76
SWAP, 127, 179
SYSTEM, 179

TAB function, 24, 179
examples of use, 26
TAN(X), 143, 179
TRACE, 60, 179
display, 61
program, 62
Triangle, area calculation program, 48
Trigonometric functions, 103
program examples, 103–106
TROFF, 180
TRON, 60, 180
display, 61

UNBREAK, 180
UNFOLLOW, 180
UNTRACE, 62, 180
Upper-case usage, 52
User-defined functions, 107
program examples, 108–109

VAL, 180
Variables, 50
integer, 51
real or numeric, 50
string, 51
Volume calculation (program), 68

WHILE ... WEND loop, 85, 180
examples of use, 86–87
flowchart, 86
WIDTH, 180
Windows:
clearing, 23
CLEARW command, 23
COMMAND, 19
EDIT, 19
FULLW command, 24
initial Desktop display, 19
LIST, 19
making full size, 23
OUTPUT, 19
sizing and moving, 19
WRITE, 181

XOR logic expression, 87

Sample programs

Area of a triangle, 48
Arrays, 121, 122, 123, 124
ASCII character set, 112
Average of a list of values, 70, 74
Bar charts, drawing, 96
Bill checking, 144
Boolean operators, use of, 89
Bubble-sort routine, 126, 130
Capital city display, 40
Compound interest calculation, 84
Concentric circles, 83
Conversion of units, 138
Cosine rule example, 105
Currency conversion, 141
Cylinder in perspective, 49
Displaying initials, 47
English–French test, 125
Finding addresses, 79
FOR loop, use of, 80
GOSUB, use of for drawing bar charts, 96
GOTO and GOTOXY, use of, 91, 92
Hexadecimal numbers, 114
IF ... THEN, use of, 79
IF ... THEN ... ELSE, use of, 79

Income tax calculations, 147
Inputting string variables, 68
Inputting values, 67
Interest on lump sum investments, 149
Interest on regular savings, 148
LEN, use of, 111
LEFT$, use of, 110
Listing costs, 71
Loan repayment, 150
Loop with step, 82
Loop with tabular display, 82
MID$, use of, 111
Multiple copies, 142
Names and phone numbers, 46
Natural and common logarithms, 107
ON ... GOTO, use of, 93
Phone number listing, 135
Random number check, 102
Random number generation, 101
Random numbers, simple game using, 102
READ–DATA, statements, 70
Repeated conversions, 93
RESTORE, use of, 73
RIGHT$, use of, 110
Running total and average, 87
Sequential files, use of, 135
Simulation example, 90
Sine rule example, 105
Sorting:
  alphabetically, 130
  numbers in order, 127
Squares and cubes, 66
Standard letters, 143
STOP and CONtinue, use of, 72
String variables, 51, 52
  inputting, 68
Temperature conversion, 67
Test TRACE, 62
Trigonometrical function, 103
User-defined functions, 108, 109
VAL and STR$, use of, 111
Volume of cylinder, 68
WHILE ... WEND, use of, 86
Weight conversion, 67
Writing to and reading, 133

USING ST BASIC ON THE ATARI ST

Dick Meadows

About the Atari ST Series
The Atari ST is one of the most significant new computers to be launched in recent years. Its performance and technical specifications are outstanding and comparable to machines several times the price. Moreover, it's a machine which appeals to the hobbyist and businessman alike with a large amount of available software which allows it to be used extensively in the home or the office.

This series of books has been designed to provide the Atari ST owner with practical, down-to-earth information about his computer – from the very introductory level through to advanced programming techniques and professional business uses.

About this book
The aim of this book is to explain Atari BASIC to both the newcomer to programming and the programmer unfamiliar with Atari BASIC. The book covers features of BASIC, ideas and methods for programming, beginning to write programs, inputting and outputting information, making decisions, repetition, special functions, processing strings, using arrays and procedures and user defined functions. The book includes many worked examples.

About the author
Dick Meadows is Head of Electronics and Communications Engineering at the Polytechnic of North London. He is an acknowledged author with over 20 books to his credit and a wide variety of research papers and articles. He is currently a consultant to the microelectronics industry and has been a visiting UNESCO Professor at the universities in Brazil and Pakistan.

Other titles in the series
Business Applications with Atari ST
Introducing the Atari ST
Using GEM on the Atari ST
Using Databases on the Atari ST
Using Graphics on the Atari ST
Advanced User's Guide to the Atari ST
Practical LOGO on the Atari ST
Using LOGO on the Atari ST

£7.95

GLENTOP
PUBLISHERS ■ LIMITED

Glentop Publishers Ltd
Standfast House
Bath Place
High Street Barnet
Herts EN5 5XE

ISBN 1-85181-051-X

9 781851 810512

GLENTOP PUBLISHERS LIMITED